Open Source Testing Tools

eGuide

TECHWELL™

If you're exploring your tooling options for test automation, it doesn't have to mean a suite of costly proprietary tools. Plenty of open source testing tools have become viable options. This eGuide explains why you should consider open source tools for your testing needs—including for the specific situations of testing in production, DevOps, accessibility testing, and security testing—as well as considerations you should take into account when selecting your tools.

In this Open Source Testing Tools eGuide

Why Selenium Should Be Your UI Test Tool
Selecting a testing tool is hard work. If you look on vendor websites, you'll get marketing material promising the world. If you look on forums, you'll mostly get people trying to solve their own problems. Justin Rohrman tells you why you might choose Selenium as your UI testing tool, based on real experience with real software projects—rather than a marketing page.

Solving Production Issues Using Testing Tools
Standard web-monitoring tools can ping webpages and verify that they're responding, but they don't alert you to an issue. But you can use the technology in load testing to monitor your sites by running an interactive script that can detect issues and generate emails as needed. It runs constantly like a silent sentry, never sleeping or taking a vacation, improving your sites' reliability.

Why You Need Continuous Testing in DevOps
DevOps is more than adopting the right set of tools; it's a cultural shift that incorporates testing at each stage of the agile project lifecycle. Continuous testing is key to unlocking this culture change because it weaves testing activities into every part of the software design, development, and deployment processes, which helps everyone involved communicate more, collaborate better, and innovate faster.

What Testers Need in Their Accessibility Testing Toolkits
A software tester's accessibility testing toolkit should contain various tools, both to help testers "walk in the shoes" of their users and to quickly flag obvious problems and expose accessibility features (or a lack of them). High performance is only achievable with human skill, but these tools will help you uncover potential issues and make your product a better user experience for a wider audience.

Using Open Source Tools for Security Testing
Performing a series of security tests before deployment of your application has become paramount. But that doesn't have to mean a suite of costly tools. Plenty of open source security testing tools have become viable options. Here's why you should consider open source tools for your different types of security testing.

How Testers Can Use Docker to Shift Left and Automate Deployments
Docker has several advantages over virtual machines: It's easier to deal with, starts up faster, and requires fewer resources. Using Docker also can give testers more confidence in their releases. Developers use the same environment that will be used in production, which streamlines code delivery and shifts QA left.

Balancing Process and Tools
The limits of a tool may lead us to realize that we are not working as effectively as we can, and often, changing a tool is part of the solution. But there are good and bad ways to select a tool and how you use it. In particular there are risks when you focus first on tools before considering the problem.

Insight from the Industry

Additional Resources

Why Selenium Should Be Your UI Test Tool

By Justin Rohrman

Selecting a testing tool is hard work. If you look on vendor websites, you'll get marketing material promising the world and then a call from their sales staff. If you look on forums, you'll mostly get people trying to solve problems relating to their product: "Why won't SuperUITester++ click Save on my Flash application when there are 4 other modals open?"

Let's take a look at why you might choose Selenium as your UI testing tool with information based on real experience with real software projects—rather than a marketing page.

Why the UI

There are lots of different places you can begin testing software. I like to think of them as different layers in a product, just like sedimentary layers archaeologists dig through when unearthing ancient cities. You want to use the most appropriate tool for each layer of software you are looking at: smaller units, seeing how methods work together, fully formed APIs, simple scenarios in the UI, and complex product usage. All the layers are important; the question is how much of each you want.

The API and everything below that will give you a feel for code quality and some basic functionality. Testing the UI will help you know things from a different perspective: the user's.

Why Selenium

The WebDriver object triggers real events in the browser: mouse clicks, button clicks, entering text, and events from the keyboard. Think of each step as a building block. Stacked together, they can enable a technical team to do some powerful things. Here are a few of the more common reasons for using WebDriver.

Building Automated Checks

Probably the most common reason people elect to use the Selenium suite of tools is to drive a specific set of commands and check status—to see if the user is logged in, if the book goes into the shopping cart, or if a transaction processes.

This includes checks to make sure buttons or labels are present on a page, data that you created saved correctly, procedural aspects

of the software under test work accurately, or any number of other attributes of your product.

Put simply, you'll use WebDriver to ask questions you would normally be able to answer with a yes or a no, and WebDriver will kick out a report with the number of assertions, the ones passing and failing, and where the errors occur.

Once you have finished building your checks, you can run them once or many times and use them as a tool to discover problems, or even unexpected changes, between builds.

The questions you ask in the scripts are binary, but usually the process of writing and running them a few times will uncover important bugs that the scripts aren't even looking for.

Taking Over Long-Running Tasks

Repetitive and very long tasks are expensive (in terms of time, which means money) and can be difficult for people to get through. Humans lose focus and concentration over time or lose interest in the goal, which leads to mistakes. Scripts don't make mistakes; they do the exact same thing each time. Normally, when tasks like this are needed, the most junior person on the team is given instruction, access to the hardware they need, and the time to complete the task. This unloads the work from more senior people, but some people still occasionally end up doing work that a program may be better suited for.

If you need to create hundreds of slightly different users or gigabytes of data in a realistic way, Selenium can help.

Management Said So

Sometimes, you don't have a choice in the tooling you use. Your manager (or your manager's manager) may have heard of this new tool that can work magic. Because it's magic, of course you should be using that new tool to solve the testing bottleneck for your project.

Maybe your manager's hunch is right and Selenium can help solve some testing problems. Here are a few questions you can ask to make sure:

- **What problem are we trying to solve?**
- Are your expectations appropriate for the work this tool can facilitate?
- Is this tool appropriate for your product and how your teams work?
- Are we ready to build and support another software project?

These questions will help you figure out the motivations behind the request. Knowing that will help set reasonable expectations for what you can do and help frame your achievements as a success.

> *Scripts don't make mistakes; they do the exact same thing each time.*

What about Another Tool?

There are lots of tools available for testing the UI, including other free options besides Selenium. I'm sure you can list five without even bothering with a Google search.

So, why should you choose this tool when there are so many others available?

Growing Browser Support

The W3C WebDriver specification will be finalized soon, and with that will come WebDriver implementations from the browser makers. What that means to you is that you don't have to worry about browser upgrades breaking the tests you spent so much time making. A specification means that vendors will ship updates of their WebDriver implementation along with new browser versions.

A Large Support System

Selenium is already one of the most popular testing frameworks in the world. It also happens to be open source. This means all changes that happen in the project happen because of community involvement. This community involvement also applies to getting help for problems. Rather than purchasing maintenance contracts and submitting a support ticket, you can go to any of a number of forums, or even straight to the developers if you choose.

> *Selenium is already one of the most popular testing frameworks in the world.*

It's Fully Featured

Selenium development follows the needs and desires of the community that uses it. You can easily see the newest features available and what is coming soon by looking at the code repo. If making a feature

request isn't fast enough, you can always write the code and submit a pull request for yourself.

Something to Watch Out For

There is at least one downside to the large network of support provided by the open source community: There are lots of users all over the globe submitting feature requests and bug reports to only a few developers. The chances of getting one of your requests in the project quickly are, well, pretty small. Commercial software may not always perform better or be more fully featured, but they do have support teams at bay.

Tools and frameworks can be difficult and time-consuming to switch from once you have made an investment, so it's important to consider what is important to your project first.

I hope this outline of the benefits to Selenium helps.

Solving Production Issues Using Testing Tools

By Nels Hoenig

Our website includes a user login, and the user authentication process was stopping occasionally. Our standard web-monitoring tools could ping the home page and verify the page was responding, but being able to interact with the page was beyond the tool we had in place. We would only learn about a real issue when a customer would alert us to it. This was not acceptable; we had to find a better way.

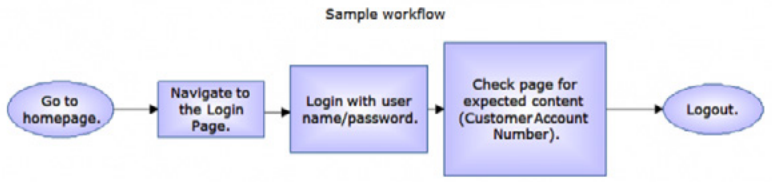
We had previously developed and executed a series of load tests using a load testing tool that allowed us to run large numbers of users against the testing website doing a number of different actions. But we needed a way to run a single user doing a simple script on a repetitive basis, twenty-four/seven, to alert us to an issue before it impacted our real customers on our production system. Our load testing software could run as a single user to do this test, but it lacked any way to generate an alert when an issue was detected.

Working with our vendor, we discovered they offered a simple solution: Use a different application to run a load test script as a single user in a repeating process and send alerts when something is wrong. We have had this process in place now for three years, and it has been a great solution. Here's how we did it.

Designing the Test

The first step is to do some business analysis to determine what to test and what a failure looks like. While similar to a load test, this test is focused on both load times for the pages and the result of the script action. You also need the ability to log in to the production system on a repeating basis using a known user/password combination.

The goal for this test is to simply verify the website is active and ready for use. Our tests did not include a transaction (sales order), but you could include this operation; it would just require more work.



These were our checks:

- Each page should load in less than 5,000 ms (five seconds)
- Each page should load correctly
- Each page should pass a text check (verification that the page loaded the expected content)

Coding the Script

Now that you have a design, you create your script.

First, we chose a valid user account that could be used in this process. (It needed to exist in production but be seen as a test account.)

We have also added monitoring of some of our intranet sites used exclusively by authorized users. This authorization is handled using SSL, so for those sites, we had to add some specialized code to support SSL and port mapping.

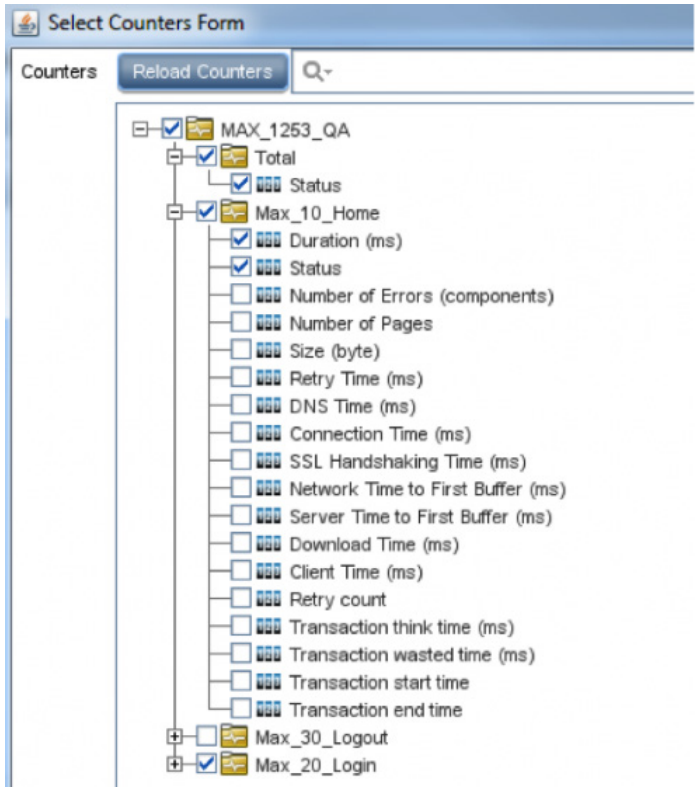
We created the script using our normal load test scripting design tool, ensuring that any special rules needed by the monitoring tool were included in the design. If you are under support, your support vendor may have some helpful information in this area.

Once you have a script designed and running correctly, now you can move it to the monitoring application.

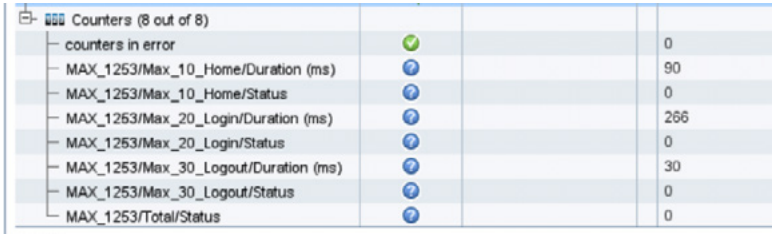
Building the Monitor

Check with your vendor about what elements from the script need to come over, as in some cases you need all runtime files and not just the script. The good news is these files are small.

When creating your monitor, first determine what web elements to track. Depending on your solution, each element you select to monitor can use a portion of your license capacity, so you may choose to limit the number of elements to monitor. In our case, we only wanted to monitor the page load result and how long it took to load, but you may choose some other elements as well, such as any of the below.

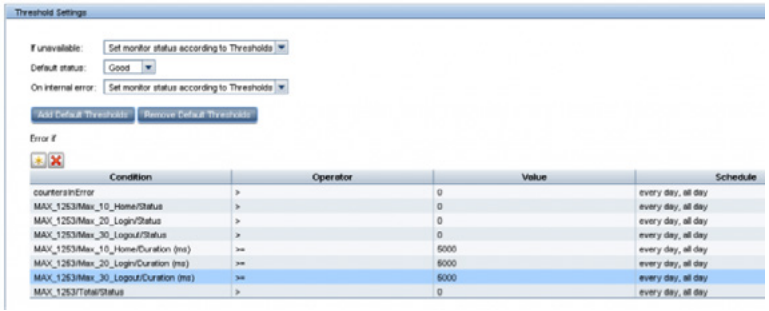


After you save the monitor, you can see the results of the initial creation on the monitor dashboard.

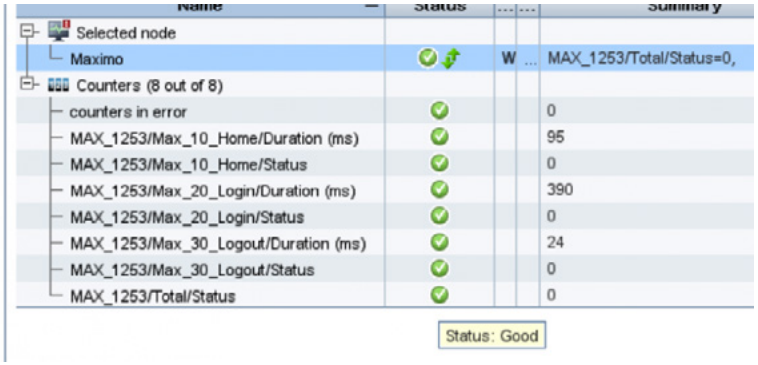


Creating Monitor Rules

Once you have the monitor built, you can create rules for what the system uses to determine a pass or fail condition. We chose to only monitor page status and load time duration.



After you have your criteria set for a pass/fail condition, the dashboard reflects the status.



You also need to decide how often you want the test to run. You can get can false alerts if a second test tries to start while the first test is still running, so we allow three minutes between each test.

Creating Alert Rules

Now that you have the tests set up and defined what a failure is, you need to set up the rules for whom to tell when a possible failure is detected.

Our tool allows more than ten different alert actions, but we chose email for our alerting process. We use a distinct email subject that reflects the website with the issue. We can send either an email or a text message, if the cell carrier supports an email address—5055551212@vtext.com, for example.

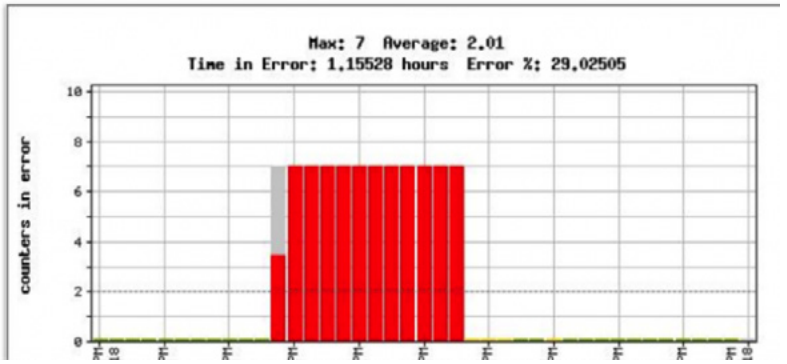
One of the things to recognize is the technology is fallible, and occasionally things break that are not on your website. To reduce the number of false alarms, we do not generate an alert unless the test fails three times in a row. We also established rules to only generate a repeat email once an hour (every twenty times), as when the ops team is dealing with an issue, they don't need a bunch of emails telling them about an issue they are already addressing.

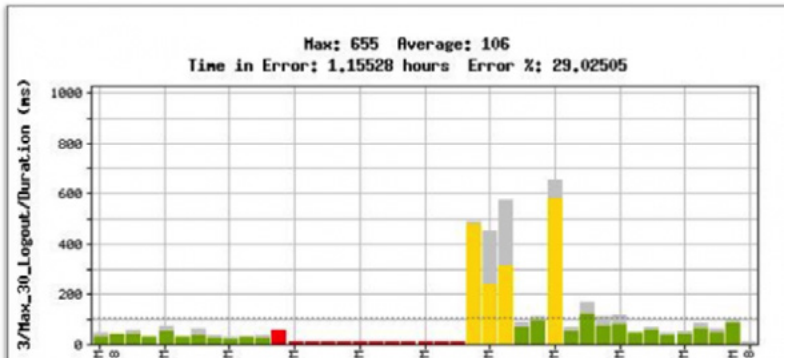
We can also go in and disable the alert as needed to prevent planned downtimes from generating false email alerts.

Generating Reports

These monitoring tools provide a nice way to examine performance over time and generate useful reports. These can be cut and pasted or exported as HTML. (I find it easier to cut and paste the relevant information rather than trying to explain all the data the report can generate.)

These are sample reports from our QA instance, of the website being down and changes in time needed for the login process:





Maintaining Reliability

Our tool also includes a dashboard view that provides a single place to quickly check all monitors and see if there is an issue with one or more of them that may indicate a bigger problem.

Initially, this tool was a bit of a hard sell to the ops team, as the idea of testing using a virtual user was a new concept to some. But now that it runs like a silent sentry, never sleeping or taking a vacation, we actually have new applications coming to us asking us to monitor their site to ensure customer reliability is maintained.

This has improved the reliability of our operation—and isn't that the role of QA?

Why You Need Continuous Testing in DevOps

By Tom Alexander

The agile process is all about using short, flexible development cycles to respond quickly to customer needs. Doing this effectively these days involves building a DevOps software pipeline in order to quickly get high-quality software into the hands of your customers and receive feedback.

Most DevOps initiatives start with the adoption of continuous integration (CI) practices, where code is continuously integrated to make sure everything works together. Developers start the CI process by checking code into a shared repository many times a day. Each check-in is verified by an automated build process and some fast-running tests, allowing teams to detect errors and conflicts as soon as possible. Regression tests are run at least each night to make sure any changes made during the day did not break something else.

After CI is performed, a continuous delivery process is followed, where the application is further tested and, once it passes all the required tests, it's available to release into production. The upside of continuous deployment is that it delivers new functionality to users within minutes, as well as instant feedback to the team that allows rapid response to customer demands. Effective testing during your continuous deployment process is critical because without it, there is a big risk of continuously releasing buggy software into production.

Don't Let Testing Practices Slow You Down

Continuous testing, which is often called shift-left testing, is an approach to software and system testing in which testing is performed earlier in the software lifecycle. The goals are finding defects earlier, increasing software quality, shortening long test cycles, and reducing the possibility of software defects making their way into production code during deployments. Continuous testing is critically important if your company is trying to use DevOps to deploy software frequently into production.

Done right, continuous testing provides fast and continuous insight into the health of the latest build of your application. This information can then be used to determine if the app is ready to progress through the delivery pipeline at any given time. Because testing begins early and is executed continuously, bugs are exposed soon after they are introduced, which reduces the time and effort needed to find and fix them. Consequently, it is possible to increase the speed

and frequency at which bug-free, high-quality software is delivered, as well as decrease technical debt.

Technical debt refers to the price organizations pay when releasing badly designed code. It's a way of calculating the cost of additional rework caused by choosing an easy and quick solution now instead of using a better, less buggy approach that would take longer. Just like financial debt, technical debt incurs interest that must be paid, such as increased maintenance, support, or legal costs. By shortening the time it takes to fix buggy software, continuous testing helps pay down your technical debt by keeping these interest costs from accruing.

DevOps is a cultural shift that promotes collaboration among all teams, including development, quality assurance, operations, and others, such as performance management, release management, and maintenance teams. Consequently, there is no single product that can be considered the definitive DevOps tool. Often, a collection of tools from a variety of vendors is used in the stages of a DevOps process. Continuous integration is often seen as the backbone of a continuous delivery pipeline, which explains the popularity of CI tools such as Jenkins and Bamboo to build, test, and deploy applications automatically when requirements change.

Companies using DevOps often ship new software into production hundreds of times every day. These companies are delivering smaller pieces of software, collaborating, and monitoring in production to create a continuous flow of code, from check-in to production. And they're using continuous testing technology to weave testing activities into every part of their software design, development, and deployment processes.

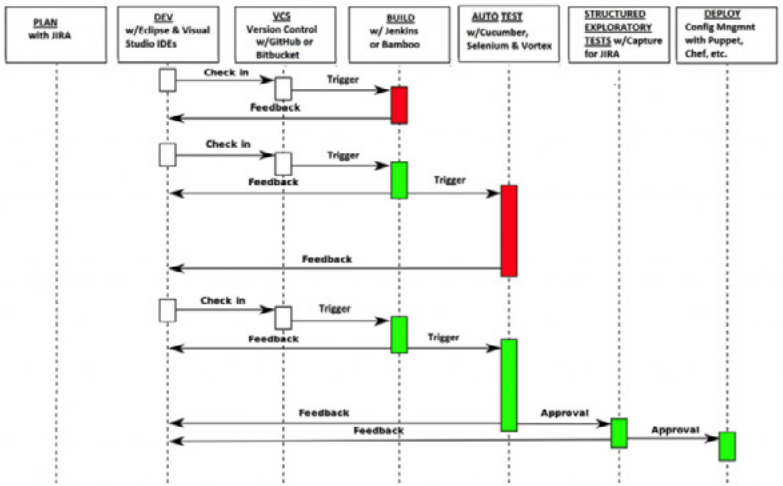
Let Tech Do the Heavy Lifting

To release high-quality code faster, your organization needs to let tech do the heavy lifting by adopting next-generation tools and practices that enable you to test early, often, automatically, and continuously.

By executing the right set of tests at the right stage of the delivery pipeline—without creating a bottleneck—these tools enforce agile principles by providing appropriate feedback at every stage of the process. This enhanced communication averts duplication of efforts and increases alignment among dev, ops, and testing teams, which will allow you to deliver software on tighter schedules.

But these streamlined schedules are only possible if test automation is seamlessly integrated into your software delivery pipeline and DevOps toolchain. Test automation works by running a large number of tests repeatedly to make sure an application doesn't break whenever new changes are introduced. Manual testers are often still involved in DevOps projects, performing testing while an automation test suite is constantly running—but their role needs to shift toward a session-based exploratory testing approach, focused on areas with the most risk or where automation is not effective.

The image below shows an example DevOps pipeline that incorporates continuous testing during check-ins, continuous integration, and continuous delivery.



A Continuous Testing DevOps Toolchain

While not an exhaustive list of all available DevOps products, here's a checklist of tools that together make up a viable continuous testing DevOps toolchain.

Planning Tools

If you're looking for a tool that makes it easy for different teams to collaborate, Jira is an agile project management tool that supports any agile methodology, be it Scrum, kanban, or your own unique flavor. From agile dashboards to reports, you can plan, track, and manage all your agile software development projects. Jira's wide range of integrations also helps you connect to almost any other tool you're likely to need.

Dev Tools: Desktop or Cloud-based IDEs

While **Eclipse** and **Visual Studio** are the most popular desktop IDEs, **Cloud9**, developed by Amazon Web Services, and **JSFiddle** lead in the cloud.

Version Control Systems (VCS)

There are several web-based hosting services for DevOps version control, including Microsoft's **GitHub**, Atlassian's **Bitbucket**, and the open source **GitLab** service. All work within standard desktop or cloud IDEs to ease the processes of source code check-in and checkout.

Build Tools

Jenkins is a CI/CD server that builds applications, runs tests automatically, and pushes code through your DevOps pipeline every time a developer checks new code into the source repository. Because of the rich ecosystem of plugins, Jenkins can be used to build, deploy, and automate almost any software project.

Bamboo is a CI/CD server from Atlassian. Like Jenkins and other CI/CD servers, Bamboo allows developers to automatically build, integrate, test, and deploy source code. Bamboo is a commercial software that is integrated and supported out of the box with other Atlassian products, such as Jira for project management and Hipchat for team communication.

Automated Testing Tools

Cucumber is a tool for specifying application features and user scenarios in plain text. Cucumber runs automated acceptance tests written in a behavior-driven development (BDD) style that encourages collaboration on software projects by writing test cases in a natural language that nonprogrammers and domain experts can read.

Selenium is a suite of different open source software tools that enable automated testing of web applications across various browsers and platforms. Most often used to create robust, browser-based regression automation suites and tests, Selenium, like Jenkins, has

> *To release high-quality code faster, your organization needs to let tech do the heavy lifting by adopting next-generation tools and practices that enable you to test early, often, automatically, and continuously.*

a rich repository of open source tools that are useful for different kinds of automation problems.

Agile teams can execute one-touch control of test automation from within the Zephyr platform with **Vortex**, Zephyr's advanced add-on that allows you to integrate with a growing suite of automated testing frameworks (including eggPlant, Cucumber, Selenium, UFT, and Tricentis) with minimal configuration. Besides being able to control the execution of thousands of automated test cases, Vortex makes it easy to automatically create test cases from test scripts and to apply insights from analytics on both automated and manual testing activities.

Session-Based Exploratory Testing

PractiTest is a test management system that supports session-based exploratory testing practices. Session-based exploratory tests are created and added to a test set during testing. These tests can be combined with other types of tests, including structured manual and automated, to maintain test suites, traceability, and test coverage.

> *Continuous testing is key to unlocking this culture change because it helps everyone involved communicate more, collaborate better, and innovate faster.*

Capture for Jira helps testers on agile projects create and record exploratory and collaborative testing sessions, which are useful for planning, executing, and tracking manual or exploratory testing. Session-based test management, a type of structured exploratory testing, is an extremely powerful way of optimizing test coverage without incurring the costs associated with writing and maintaining test cases. Like Zephyr for Jira, Capture for Jira has a deep integration with the Jira platform, allowing users to capture screenshots within browsers, record screens in Chrome, create annotations, and validate application functionality within Jira.

Deployment Tools

Longtime "movers and shakers" in the DevOps infrastructure-as-code space, **Chef** and **Puppet** are both automated configuration management and orchestration tools used to quickly spin up compute and storage instances on demand.

Test Continuously to Deliver Faster

DevOps is more than adopting the right set of tools; it's a cultural shift that incorporates testing at each stage of the agile project lifecycle. Continuous testing is key to unlocking this culture change because it helps everyone involved communicate more, collaborate better, and innovate faster.

What Testers Need in Their Accessibility Testing Toolkits

By Albert Gareev

The concept that software should be usable by the widest possible audience has been around for more than twenty years, yet for quite a while it remained out of the mainstream of testing and development efforts.

This has been changing in recent years. We have seen diversity and digital inclusion become social priorities. On top of the implied social contract, we also now have explicit legal contracts, such as Section 508 of the Rehabilitation Act in the US and Canadian provincial legislations (AODA in Ontario, Quebec Standards for Accessibility, and others), which define accessibility standards for government and public sector software. This sets a trending example for the overall market.

Just like designers, business analysts, and programmers, testers need to acquire new professional skills in the accessibility domain.

Digital accessibility refers to not only ease of use within web browsers for those with visual, auditory, or motor disabilities, but also software supported by users' assistive technologies. Using tools to identify accessibility problems is a popular approach, but it has hidden pitfalls. Accessibility at large is about human perception, cognition, and interaction, and those aspects are hardly detectable by mechanistic means. Here, let's look at accessibility testing tools by category, highlighting their advantages and risks.

Why Tools Are Essential for Accessibility Testing

Testers recognize problems with software based on their mental models, experiences, and feelings. But how can we test a product effectively on behalf someone with different perceptions and mental models?

One way is learning accessibility-specific oracles—effectively, heuristic principles—in order to be able to recognize and classify barriers that may be encountered by people with disabilities. Another important method is employing tools to alter your own perception and cognition so you can model the usage patterns of your customers.

The ways customers navigate and operate the software product may vary depending on the cognitive, motoric, and habitual characteristics of their interaction with the software. It also depends on users' knowledge of domains, experience with particular products, and skill level in using assistive technologies in particular and information technologies in general. That asks for a variety of tools to be employed by testers.

Accessibility testing tools include those specifically created to identify defects as well as regular assistive technologies used by testers in the same way as people with disabilities. Let's look at some of the most popular tools.

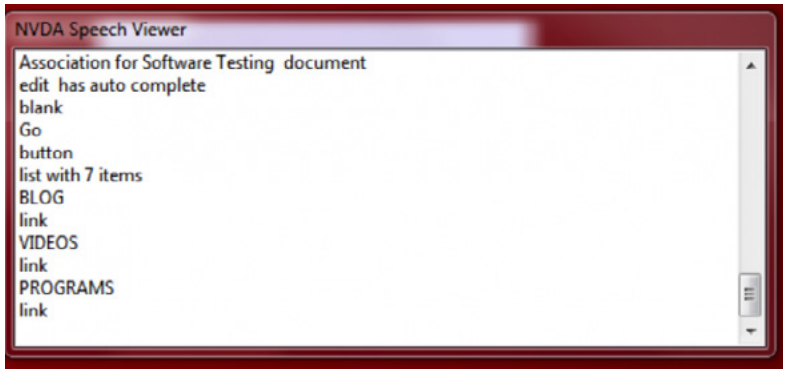
Screen Readers

People with vision impairment use screen readers, software programs that read the text displayed on a screen using a speech synthesizer or Braille display. The user can send commands by pressing different combinations of keys to instruct the speech synthesizer what to say, to read or spell a word or full screen of text, announce the location of the computer's cursor, and more.

Screen readers are available as commercial and as free or open source software (such as popular English-speaking options JAWS and NVDA), and there are also built-in options for Windows, macOS, iOS, and Android.

Screen readers are not testing tools by intent, but they are invaluable for accessibility testers to simulate their users' experience firsthand.

Some screen readers come with useful features like a speech log, which helps testers identify problems and back up bug reports. In the example below, the free and open source screen reader NVDA shows through its speech viewer that the website's dictated tab order does not match visual order—the tabs are supposed to be ordered left to right, but with keyboard-only operation, users get right to left, with "Blog" given first.



HTML-Checking Tools

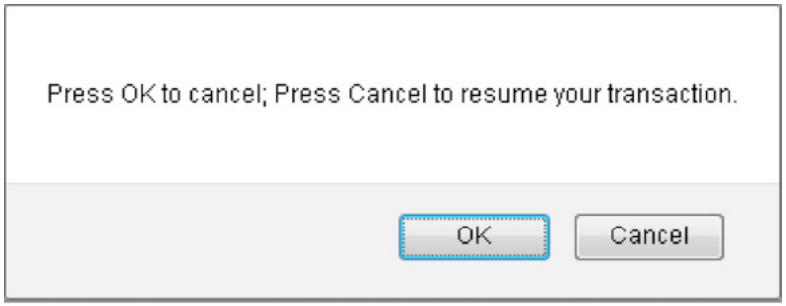
As the name suggests, HTML-checking tools scan webpages' content and check syntax against the rules encoded in them. The rules are all based on the international standard Web Content Accessibility Guidelines, or WCAG.

There's a wide variety of these tools on the market, with free and open source options as well as pricey enterprise products. They are available as standalone applications (such as SortSite), online applications (AChecker and WAVE), API integrations (Tenon), and various browser add-ons (WAVE also has a toolbar).

This class of tools is useful for a quick search of "low-hanging fruits" such as obvious problems caused by missing HTML elements needed for accessibility. For example, if an image doesn't have an associated textual alternative, or alt text, a tool will flag it.

However, the tool still requires a human to decide whether the alternative text adequately describes the image in the particular context. A tool also won't distinguish between an image used just for visual decoration, which should have empty alt text, and an illustration image that must have meaningful alt text.

In fact, anything concerning adequacy of the user interface requires human evaluation. For example, this screen shot of a modal dialog has no HTML violations, but there are obvious usability problems relating to understanding this message.



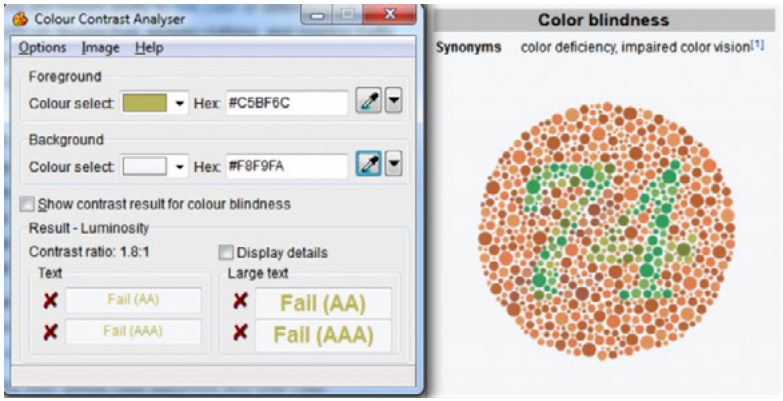
Typically, HTML-checking tools evaluate a single webpage or website. They don't have the ability to automatically navigate a web application because that requires data input and user actions. They also do not provide full coverage of all WCAG criteria and they do not catch all problems for any criteria.

Color and Contrast Checkers

Some people cannot distinguish between certain colors. They may not need special assistive technologies, but due to this difficulty, a user interface should not rely solely on color to convey information. Additionally, for people who have epilepsy that is photosensitive, flickering or flashing lights can trigger seizures, so these elements should not be used on websites, either.

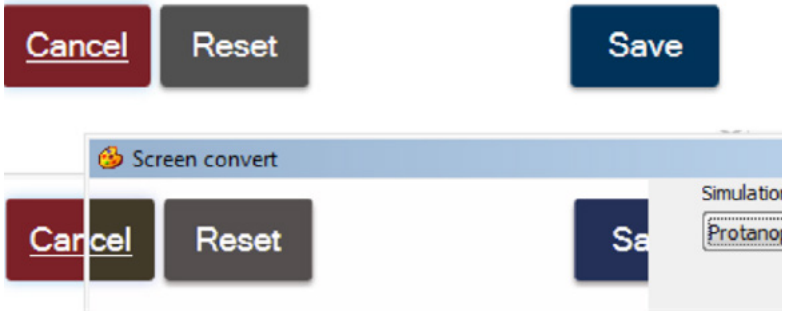
Color and contrast checkers can help detect these elements that could be problematic. As the name implies, this family of tools is used to verify color code and contrast ratio. Typically, they require a tester to manually select the color of the background and foreground in order to calculate the ratio.

In this example, the Colour Contrast Analyser tool evaluates the image from the Wikipedia page on color blindness and, predictably, shows that the text fails the tests.



> *A user interface should not rely solely on color to convey information.*

More advanced tools may include a simulation mode to give testers a firsthand perception experience, so they'll know what their users would see. Below, the same tool in simulation mode shows that the red Cancel button would be indistinguishable from the gray Reset button for people with protanopia, or red-green color blindness.



GUI Automation Tools

Regular GUI automation tools, such as Unified Functional Testing (UFT) or Selenium, can be successfully employed for regression testing with partial coverage of accessibility requirements. Because GUI automation tools do interact and navigate web applications, sometimes it is viable to incorporate custom checkpoints to overcome limitations of generic accessibility checkers. Verification of accessibility features in HTML is not different from verification of ID, labels, and other elements routinely verified via GUI automation.

Readability Analyzers

For people who have a limited ability to process and memorize information, make decisions, or pay attention for an extended period of time, overly complex language can be confusing or frustrating. This category also includes people in stressful or distracting conditions or who are not fluent in the language. To assist these users, software should present information in a clear and organized manner, remind people about important points, and allow verifying and correcting. These characteristics together are known as *readability*.

There are linguistic methods to calculate the complexity index of a written text (for example, the Gunning fog index or the Flesch–Kincaid readability test). The index estimates the years of formal education a person needs to understand the text on the first reading, and this can be tied to a level of cognitive effort. There are free online tools for text evaluation based on these and other criteria. As with other mechanistic means, you can't rely on these tools alone—a human verification is still preferable—but they are useful for a quick, rough check to flag potentially problematic language.

Ordinary Tools

People with low vision, defocused vision, or spot vision may not need to use screen readers but would still like assistance in reading text. They may use the zoom feature in a browser, reduce screen resolution, or increase font size, contrast levels, and color polarity.

People who temporarily or permanently do not have a desired control of their body, especially their arms or hands, may use a variety of physical and electronic assistive technologies, but in terms of software operation, it comes down to a keyboard-only usage pattern without the aid of a mouse.

These options do not require any special testing tools. For example, the ability to navigate through an entire web page using a keyboard only (tab and shift-tab) is a basic level of compliance. While "tabbing through," testers also should check that the focus is always visible and well distinguishable.

Other useful tools for testers already are built into the browsers. Pressing F12 opens the Inspect mode, allowing testers to check the HTML source of a UI element in question.

In the example below, alt text for the images was highlighted using the Web Developer toolbar, and we have examples of both sufficient and inappropriate textual descriptions.

Such browser tools may not specifically point to accessibility problems, but they still greatly help testers find them.

Add to Your Toolkit

A software tester's accessibility testing toolkit should contain various tools, both to help testers "walk in the shoes" of their users and experience what they experience, and to quickly flag obvious problems and expose accessibility features (or a lack of them). As in every aspect of skilled testing, high performance is only achievable with deep engagement of human skills, not a reliance on tools, but these will help you uncover potential issues and make your product a better user experience for a wider audience.

Using Open Source Tools for Security Testing

By Saurabh Hooda

Without sufficient security testing, software applications are at high risk of being hacked, phished, or attacked by computer viruses. Performing a series of security tests before deployment of your application has become paramount.

But that doesn't have to mean a suite of costly tools. Plenty of open source security testing tools have become viable options.

Open source security testing software is available online and can be downloaded from the developer community's website or from the vendor's product website for free.

Here are the different types of security tests where an open source security testing tool can help.

Vulnerability Scanning
Vulnerability scanner applications assess computers, devices, and networks to discover security weaknesses in the system. Vulnerabilities can arise from errors in programming or configurations in firewalls, routers, web servers, application servers, and so on. This scanner also provides trend analysis and countermeasures to eliminate the discovered vulnerabilities.

Penetration Testing
Also known as pen testing or ethical hacking, this is a type of testing where a cyber attack is simulated on the system. The tester acts as a real hacker and performs various testing scenarios.

Security Risk Assessment
This is a preventive approach that should be an integral part of your organization's risk management process. Security risk assessment tools provide a road map for a strong security approach for your application development and deployment environment. Security risk assessment tools can identify critical security risks in your application, define a mitigation plan for the risks , and prevent threats and vulnerabilities by implementing security testing tools.

Probably the chief benefit of opting for open source security testing tools is their cost-effectiveness. Open source tools are free downloads, so they can be installed on multiple machines at zero cost.

But another benefit are the periodic upgrades. Most open source tools have an active developer community, so you get regular enhancements to the tools when the products evolve and new features and bug fixes are rolled out. However, keep in mind that not all open source tools are backed by large organizations, so you may run into issues in the long term.

Finally, just like with a proprietary tool, it'sall yours. Most open source software not only allows customization to suit your application's requirements, but also enhancements for efficiency and new features. Effectively, you own the tool and can play around with it.

The development community for open source security testing tools works to deliver scalable and relevant tools to the software industry, along with any patches and improvements, at no cost to you. A proactive approach to the evaluation, selection, and deployment of open source security testing tools will mitigate the risks of exposing your software and your system environment to malicious hackers and intruders.

How Testers Can Use Docker to Shift Left and Automate Deployments

By Artem Golubev

Docker is an increasingly popular lightweight virtualization system. It has several advantages over virtual machines, such as having a declarative description of the file system, and it's easier to deal with, starts up faster, and requires fewer resources in general.

In particular, the layered file system lets you deliver new updates to data centers and get updates from developers faster, and being lightweight means it is much easier and more efficient to run locally. There are images as small as 2 Mb—you'll never find a VM image that small.

It's also easier for developers and ops to create lightweight OS images with software already set up, and they can be run exactly the same way on both the developer's machine and the test and production environments.

There are multiple ways Docker can be used. I'll dig deeper into how it can help with testing in continuous delivery and integration (CD and CI) in certain cases.

The increasingly popular way of setting up a development process is a pull request-based workflow popularized by GitHub:

1. A developer writes code and creates a pull request, adding their code to the main code
2. Other developers can now review the code, and it can also be deployed
3. Once review and testing is complete, code is merged to the main code
4. The new code can be directly deployed to the production environment (full CD) or to a test environment (CI).

When Docker is added to this workflow, a deployable Docker image is created during step one, which is the same image that eventually will go to production. It also usually can be pulled to a developer's machine for debugging, to avoid the "But it works on my machine" issue. In this workflow, not only can all end-to-end tests be run on the pull request code under test, but they also can be deployed and tested by QA.

Docker Compose is a way to join several images to work in tandem, such as a web server and database in one compose. Using Docker and Docker Compose, it is possible to deploy the pull request's code on a completely fresh environment with a fresh database, eliminating issues with contaminated data.

The alternative, more complex way to do the same thing would be by using Kubernetes, which is designed to deploy sets of images (similar to Compose) to production or test environments.

The ability for developers to use exactly the same environment that will be used in production helps to not only eliminate and debug environment issues, but also simplify and streamline delivery of the code to both test environment and production. The same image can be used on a developer machine as on a test environment and, once tested, can be delivered to production.

This also enables deploying those environments on demand and working with each image (potentially related to a code branch) separately, isolating any issues. This constitutes an effective shift left, which allows QA and developers to work much more effectively.

Balancing Process and Tools

By Steve Berczuk

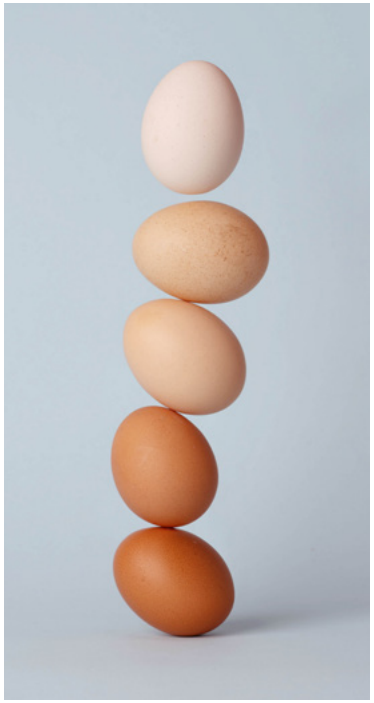
It's the rare team that doesn't need to change something about its processes or tooling. The limits of a tool may lead us to realize that we are not working as effectively as we can, and often, changing a tool is part of the solution. But there are good and bad ways to select a tool and how you use it. In particular there are risks when you focus first on tools before considering the problem.

There are many reasons you might start with tools, some valid, some not. Constraints such as cost or existing licensing arrangements might limit your choices. Starting with tools is also a good way to feel like you are making concrete progress.

But be mindful of making decisions based on hype. A popular tool, or one endorsed or developed by a successful company, is worth a look for many reasons, but it's important to consider whether the tool actually fits your needs. Some oft-mentioned tools are useful if you are working on the same problems as other companies, but you may not be.

Like other technical choices, the risk associated with tool selection can be mitigated by experiments, but a successful experiment depends on having the right evaluation criteria. A good way to understand your problem and requirements is to think about your current process and the process you want to have. You can then evaluate how well a proposed tool supports the new process.

This sounds obvious, but teams often assume that they have a clear understanding of their problem and don't challenge that assumption. Sometimes a brief analysis of the problem may lead to a simpler solution than you thought. Sometimes even doing nothing may be the right approach.



Being inflexible with your process is another risk. You may find a tool that is close to what you need but which does not do things exactly as you wish. In these cases you may be tempted to extend the tool in some non-standard way. If the tool is widely used to solve a fairly standard problem, it can be worthwhile to reconsider your approach—your problem may not be as unique as you think. Blindly extending or adapting a tool to fit your nonstandard needs can lead to extra work and, eventually, avoidable tech debt.

There are many possible inputs into your decision making process. The key to a successful decision is to ground it in the technical, process, and business needs of your application, rather than hype or unsupported assumptions. Examine whether you can fit your process into the idioms your tool supports instead of adding complexity. If you find yourself making many customizations, consider whether your problem really is *that* different from others.

By starting with your process goals first, you can do a better job of evaluating and selecting tools. This will lead to better decisions and, in the end, lower costs.

Insight from the Industry

"Open source tools are vital to the way that many of us do our jobs, whether we know it or not. The testing industry has become somewhat stagnant in regards to the types of tools that we use. ... Selenium's rapid rise is indicative of a need for simple, affordable tools that can easily be shaped and built upon to fit each company's unique needs—which, of course, is what open source is all about."
—Mike Sparks

"I've been doing automation for a long time. Open source really gets me excited for a lot of reasons. Software development as a whole is moving more and more toward open source—not just test automation, but software development as a whole. I really like the idea of having your ideas being shared and building out features that everyone can use."
—David Dang

"What I do love about open source is that you, as a technical person, have an opportunity to give your feedback. You have an opportunity to submit your own code and your own revisions and feel as if you're a part of something bigger and greater."
—Stacy Kirk

"The software development lifecycle is measured in hours or minutes these days as compared to months or years a decade ago. For Dotcom 1.0, the tools were rudimentary at best. Now we have open source tools and frameworks that solve a lot of the problems of getting a web or mobile application up and running. As the tools evolved, so did the development process."
—Michael Nauman

"Certainly open source tools are a lot more prevalent, and if you're only starting in automation, that's a very good way to start—is just to experiment by using a few open source tools and see what you really need, see what you want, see what works for you, even if you're planning to go for a commercial tool."
—Dorothy Graham

"Obviously, the first part of that change [when evolving your testing] is moving to more open source tools—moving away from some of the big third-party automation tools and adopting the open source like Selenium and Cucumber, and Espresso, Appium—really using them, because this is how you're going to be able to automate in real time and keep pace with developers."
—Adam Auerbach

"It's very hard these days with all the speed that everybody needs and the quality to really stick to one single end-to-end platform. That is just like, we don't find customers that have environments that are unified in one single platform. The ones that are, are trying to get out of that simply because, like we were saying, things change so fast and there's always something better to explore. Right now what we're focusing on is really connecting and interfacing with the open source and commercial side of things."
—Alex Martins

"[Testers] want all these tools, whatever tools they use, whether it's open source, commercial, to be seamless in terms of the process. They're using the best of their time for the things that they do well, which is in their brains."
—Julie Gardiner

> *Open source tools are vital to the way that many of us do our jobs, whether we know it or not.*

Additional Resources

MORE INFORMATION FOR SOFTWARE PROFESSIONALS

STICKYMINDS™ A TECHWELL COMMUNITY

StickyMinds is home to thousands of software testing resources, including informative articles about all aspects of testing, *Better Software* magazine articles, and interviews with industry notables. StickyMinds offers how-to advice and views on the latest ideas and practices from experienced software professionals and thought leaders. Join the community to gain access to exclusive members-only content such as conference presentations, Q&A discussions, a weekly newsletter, and more.

CLICK HERE

NARROW YOUR SEARCH TO A SPECIFIC TYPE OF RESOURCE:

StickyMinds Articles

StickyMinds articles cover a wide range of software testing topics, including open source testing tools, test automation, test management, test design techniques, agile testing, test process improvement, and much more. **Click here** to read articles about open source resources on StickyMinds.

Interviews

Each year, TechWell interviews dozens of software professionals, including well-known thought leaders, seasoned practitioners, and respected conference speakers. **Click here** to read, listen to, and watch interviews with testing experts about open source options.

TechWell Insights

Find stories about testing, security, open source tools, agile, DevOps, and more, all written by software industry professionals. New stories are added each week, so **click here** to read the latest—or sign up to receive the weekly newsletter roundup of the newest stories.

TechWell Conference Presentations

Couldn't make it to a TechWell conference to sharpen your testing and development knowledge? TechWell conference presentations are available to StickyMinds members for free soon after conferences end. **Click here** to join StickyMinds and access conference presentations related to open source tools and implementation.

STAR Conferences

The STAR conferences feature keynote presentations, tutorials, and classes covering test techniques, performance testing, test automation, agile testing, mobile testing, test management, and more. Learn from experts in the field and network with your peers to get the most immersive conference experience possible. **Click here** to learn more.

Agile + DevOps Conferences

Explore keynotes, tutorials, and classes covering the entire agile development lifecycle and the most cutting-edge DevOps principles. These conferences give you the methods, process, technology, tools, and leadership ideas to deliver software with greater speed and agility while meeting quality and security demands. **Click here** to learn more.