Sencha

The Evolving Landscape of Web Application Testing

Executive Summary

Organizations are under pressure to deliver increasingly sophisticated, high-quality applications to their customers faster than ever. However, compressed delivery schedules, constrained resources, and increased complexity hinder their ability to deliver on those objectives. Increasingly, successful companies are transforming their approach to testing by shifting it earlier in the development cycle, investing in automation, and embracing modern web technology to gain a competitive advantage and deliver a great customer experience.

Introduction

Automated testing is a must-have tool for any organization. Continuous deployment and delivery can be achieved only by automating repetitive tests. And, automated testing has become the obvious choice for addressing the fragmented browser and mobile device market. Businesses that invest time in building robust automated tests are able to reduce the software development cycle considerably over the long run. The continuous feedback loop helps the organization to improve the reliability and overall quality of their releases.

Web Application Testing History

Landscape

A decade ago businesses only had to worry about one browser or maybe two. Testing desktop-based applications was the top priority, and browser testing was usually a last minute item on the release checklist. As more business-critical applications became web-based, organizations faced increasing risk due to inadequate testing. Businesses that used open source test frameworks tended to use manual testing that not only slowed their go-to-market strategy but also provided very little test coverage. Today, web applications built using JavaScript frameworks, such as Ext JS, can be rendered on desktop, tablets, and smartphones, which significantly increases the testing combinations. Many companies still use legacy browsers whereas consumers adopt the latest browsers and mobile technologies much faster – fueling more fragmentation and testing complexity. Multiple active versions of browsers, automatic updates by browser vendors, and mobile operating systems make it impossible to achieve full test coverage through manual testing or less efficient test frameworks.

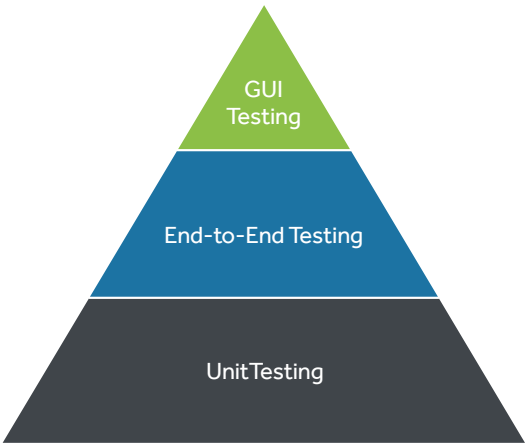
Skills

Organizations who lack the required manpower for manual testing have resorted to building custom test frameworks using open source technologies. The downside to this approach is that these frameworks don't typically scale beyond a proof of concept stage, and the ongoing management and maintenance of the custom framework can be costly. Using expert resources in test framework development and maintenance severely affects developer productivity, as test teams often seek the help of developers to tweak or fix test frameworks when third party upgrades or dependencies go wrong. Software organizations are spending a huge amount of time maintaining test frameworks rather than writing good quality tests.

Options

Development of open source test frameworks has taken off in the last decade. There are more than 35 test frameworks for JavaScript unit testing, and most of them focus on a specific testing need. A complete testing solution requires multiple specialized tools and supporting third-party libraries. Development teams need to invest a lot of effort and time assembling the right tools and maintaining the overall solution. Commercial solutions focused on a complete strategy of unit and end-to-end testing can greatly reduce the burden shouldered by development organizations.

JavaScript Testing



Unit vs. End-to-End Testing

JavaScript has been widely adopted as a first-class programming language. Frameworks such as Ext JS have provided a way for customers to write rich web applications using JavaScript. Unit testing for JavaScript has not evolved as quickly as the programming language itself. Developers are still finding it hard to isolate the smallest component and test it as an individual function. Development teams stay away from unit testing due to the lack of best practices and tools that enable them to perform tests quickly and efficiently.

This forces teams to rely heavily on end-to-end functional testing that is performed at the browser level. Many test automation teams have resorted to creating test frameworks using Selenium. Using best practices, such as page object methods, Selenium provides a structured way of writing tests, and WebDriver has lessened the burden of invoking or navigating the browser the way an end user would. In the web testing space, Selenium has largely served as the open source alternative for commercial tools.

JavaScript-based test frameworks, such as Jasmine, adopt a behavior-driven methodology in which tests are written in English language instead of writing them programmatically like in Selenium. Tests written in Jasmine are

easy to understand and have built-in "matchers" or "expect" statements that allow test authors to easily define the behavior of the test. The Jasmine framework and WebDriver can be used together to run tests on multiple browsers simultaneously – making it a very powerful alternative.

How to "Unit" Test

Unit testing is done by testing the smallest component or unit in isolation. There is often no DOM involvement at this point. This type of low-level testing occurs before the application is launched in the browser and is typically performed by the developer who built that piece of functionality in the application. The developer writes a unit test specifying the various conditions that have to be met to pass the test. The individual component or piece of code could just be a simple mathematical calculation or an email compose window with two text boxes that accept specific input values. In order to test the email compose window with dummy data, the developer can create mocks to act as an integration point and tear them down once the test is completed.

Code Coverage

While unit testing identifies the flaws in the program logic, code coverage highlights which code is exercised using the current test suite. Analysis done at a function/branch/line level helps a developer write tests for a specific area of the code that is not covered by the tests. This ensures that the developer understands which code is and is not being tested. Business stakeholders also gain insight into how completely application functionality is being tested. While it is not always practical to achieve 100% testing coverage, code coverage metrics provide insight into what is not being tested at the unit test level, so it can be targeted during end-to-end automated testing or exploratory user acceptance testing.

Browser/DOM Testing

Black Box Testing

Black Box Testing is an essential component of Quality Assurance. This is the final step before the application is deployed into production. Application features are tested from the perspective of a user, including new and existing features as well as testing for regressions. Black box testing addresses gaps in unit testing and potential differences between what has been developed compared to application requirements.

The test automation engineer doesn't know what's happening under the hood, as there is no access to the code at this point. Developers often avoid elaborate unit tests because they would take too much time to write and execute, potentially jeopardizing the release schedule. While organizations are actively trying to unit test as much as possible, they can't rely solely on one type of testing. After all the systems are integrated, functional testing is absolutely necessary for organizations that have many upstream and downstream applications. Automating end-to-end application testing (for one or many integrated applications) ensures that businesses can go to market with more confidence and at a much faster pace.

Locator Strategy

Locator strategies help with identifying a component or an element in the web application, typically for purposes of manipulating or querying that item during testing. Every element has an ID or a name associated with it that is assigned by the developer or, in the case of IDs, dynamically generated. Identifying an element using a name or an ID, that is prone to change with every build, has proven to be very unstable. For example, consider this scenario:

- Developer names a button "Submit"
- Test automation engineer uses the "Submit" name-label to create the test case
- During the next build, Developer changes the label to "OK"

This seemingly innocuous change causes the test case to fail because it is looking for the text "Submit" but it has been changed to "OK." If the test automation engineer has used the same locator strategy in 100 different test cases, they will all have to be changed.

To prevent such brittleness in tests, test writers need to carefully choose a stable locator strategy. When deciding on a good strategy, it's crucial to consider the dynamic nature of the application and choose one or a mix of strategies to identify a component. Using a component query along with a DOM query or an XPath provides considerable protection against the changing nature of the application. Testing the locator strategies multiple times before using them in tests is highly recommended. Also, it's very helpful if test automation engineers work closely with developers to understand which properties of components and elements are stable.

Test Design

A clear test strategy is critical to ensuring the quality of an application. Good testing cannot be achieved simply by running a large number of test cases or having good test programmers. Test design is at the core of creating good tests. Clear separation of testing logic and UI verifications helps to identify problem areas in the code. This ensures that business-critical functions are performing well, and the UI is rendering all of the elements seamlessly. Separating logical tests from UI verification also helps with better reporting. A test automation engineer or manager can quickly scan through all critical functionalities both from a logic and UI perspective. Maintenance of large numbers of test cases becomes very easy if tests are clearly designed. By boxing tests together, testers can build tests for features that they are responsible for and maintain them over the long run. It becomes easy to identify regression tests that need to be run based on new feature development. Also, obsolete or redundant tests can easily be isolated without impacting other test cases.

Finally, good tests cannot be created by focusing on test programming. Test strategies should be formalized based on business requirements and technical architecture. Test programming is a great aid for writing good tests. Once the tests are defined, a minimalistic programming strategy will help with creating clean tests.

Role of Testing in DevOps

Continuous Integration/Delivery

DevOps teams focus on application development, deployment, and the infrastructure needed for continuous integration and deployment. Automated testing is a key element of this system. The Dev and Ops groups have many tools that help with automation such as the source control mechanism, build process, and deployment to servers. If testing is not automated, it creates a huge gap in the process. Many organizations struggle to cross the last mile of continuous integration and deployment, so test automation solutions that address this critical area are particularly valuable.

Automated Testing

The goal of using automated testing should be to minimize the time spent in performing repetitive monotonous tasks such as isolating test scenarios that have to be performed every time the application is modified. Once these scenarios are identified, the test teams should ensure they have the right test strategy in place. Along with the test scenarios, the strategy should include the required data and infrastructure. The test data is an essential aspect of automated testing because poor data choices could lead to false positives or negatives during testing. Infrastructure is another key part of automated testing. Test teams have to choose between virtual machines and browser farms (cloud providers such as Sauce Labs) based on their security protocols. Understanding internal infrastructure capabilities ahead of time helps test teams choose the right automation solution.

Results Reporting

Most teams spend a lot of time designing the test strategy and on many other test activities, but they seldom think about results reporting and analysis upfront. When it comes to reporting, test teams resort to outputs from the continuous integration process. This approach works well for small teams with few test cases but not for large teams with tens of thousands of test cases. Depending on text-based results files forces teams to create separate modules to parse and generate reports in HTML or other preferred formats. A good test automation solution should have a concise way of collecting, aggregating, and presenting the right results. It should also have a robust archiving mechanism along with the ability to retrieve reports in many different formats, as many organizations have to store test results for audit purposes and legal reasons.

Cross-browser testing of web applications is a clear mandate for many organizations. Linear generation of results would require a lot of effort from testers to sift through one log file per browser and conclude if the application can be deployed to production. A matrix presentation of results enables testers to identify the failures across multiple browsers quickly and at a glance. This is a critical capability when simultaneously testing multiple browsers with varying browser versions on multiple operating systems.

Automated Testing

The Myths and Truths

	Myth	Truth
Process	Automated testing just requires creation of appropriate tests. Once tests are created, they no longer require attention.	Automated testing is an ongoing process. The system needs regular maintenance and test updates based on changes to the Application Under Test.
Collaboration	A software developer or test automation engineer can create tests on their own.	A robust testing regimen requires a collaborative effort among all stakeholders.
Fitness for purpose	All automated testing tools are well suited for web application testing.	Teams can benefit from significant boosts in effectiveness and efficiency by using automated testing tools that specialize in the web development frameworks they use.
Time to value	Teams will quickly realize value once they start doing automated testing.	Teams need to balance short-term benefits with long-term goals. Most of the value from automated testing will be realized over the long run.
Expertise	Effective automated testing can be achieved by adding skilled test programmers to the development team.	Effective automated testing requires the support of developers, IT staff, testers and other stakeholders. Teams need to consider team member skills, monitoring/reporting requirements, and infrastructure needs.
Breadth	A single testing tool or solution is sufficient for all testing needs across web, mobile, and desktop-based applications.	A multi-tool strategy helps to focus the test plan. Specialized testing tools can help solve testing problems easier than a universal tool.

When designing a test strategy, teams should consider the future changes to the application under test and their impact on the tests. A good design allows flexibility for test teams to accommodate any radical changes to the application. Often, teams fail in this area due to a narrow focus on short-term ROI and the inability to invest time upfront to build a good strategy. Many constraints such as software release pressure, lack of resources, and short deadlines from senior management can lead to test strategies that are not scalable over the long run and end up being scrapped. Teams should appoint a technical test architect who can take charge of the test strategy and think about both the present and future state of applications. By segregating functional and UI verifications as well as different functional areas of the application, the automation solution has room to grow as the application changes in different areas. By doing so, the test team does not have to change many tests to accommodate a simple functional change in the application. A good test automation tool not only provides a way to write test code but also acts as a platform for creating and maintaining a good test strategy. By adopting a process-driven approach, such as using a Behavior Driven Design (BDD) syntax, the tests can be broken down by functional and UI verifications.

ROI

As stated above, short-term ROI for automated tests is a myth. Businesses that start a multi-million dollar project typically have to wait at least a year to realize some benefit from it. A similar rationale can be applied to test automation projects. An automation project goes through many phases:

- Determine test strategy
- Create test design
- Prepare data
- Set up infrastructure
- Write test code
- Integrate with CI integration
- Run unattended tests

To achieve success in an automation project, test teams have to be meticulous about each one of these stages, which can take anywhere from days to weeks to get them right. Teams have to stay focused at every stage and make sure they have all the right elements in place. Rushing through any of these stages can hurt the team's success over the long run. Therefore, it's prudent to not calculate ROI for the initial couple of cycles in a test automation project. By doing so, teams can get all the "wrongs" right and put a robust system in place.

Test Frameworks

Open Source

There are many reasons why teams choose open source test frameworks: flexibility, following the latest trends, and so forth. Because most teams understand the importance of testing, they typically do not need to justify the testing effort itself. Rather, they need to get management support for an evaluation and selection effort. Depending on the organization's culture, that support might be sought before any investigation is performed, or development teams might perform their own investigation before presenting a proposal to management.

Teams often start by considering free and open source testing tools because they have minimal budget to get started. It's important to remember that this does not mean that adopting such tools will have no cost. Teams often forget about the cost of building everything from scratch, maintaining test framework updates, and depending on third-party forums for support. The costs of these dependencies are intangible, so teams tend to leave them out of their decision-making. Over the long term, these costs often dominate the overall cost of the testing effort and can become so large that the testing effort will be entirely restarted with different tools. As such, these costs must be considered in the business justification for any proposed testing solution.

Open source test frameworks have matured over the years and are definitely more stable than the frameworks available a decade ago. Still, teams are finding it extremely difficult to create and maintain test automation by leveraging open source frameworks. The fast-changing landscape of the open source community and the abandonment of frameworks that were "hot" a year ago make it extremely hard for test teams to keep up. Frameworks can have substantial changes when a new major release comes out. The open source community tries hard to minimize the impact, but it's impossible to support every user, as there are no dedicated support teams or service level agreements for open source projects. These changes may render all tests useless that were written using a previous framework, or teams are stuck using older versions of the framework. If teams don't upgrade to the latest versions of these frameworks, they get left behind as open source frameworks undergo changes very rapidly. In addition to this issue, teams have to retain skilled test programmers, as it is very hard to find a good developer who will maintain the test framework.

Commercial

There are commercial solutions for automated testing, but many of these tools take on everything from web to desktop to mobile. In order to meet the testing needs of a broad range of application platforms, these tools are overloaded with many features that create a heavy product footprint.

While many of these products are able to test the DOM/Browser level, users must look at comprehensive solutions that not only address browser-level testing but also address the unit-testing level. Organizations that are leveraging JavaScript frameworks, such as Ext JS, require special capabilities in their test product that will allow them to easily interact with their application while creating tests.

Unlike many open source solutions, commercial solutions often try to address common business and process problems as well. For example, they may offer easy integration with continuous integration tools, browser farms, and reporting dashboards.

How to Choose

There is no single choice that is right for every situation. Considering the specific needs of each team and project will help guide companies through the maze of open source and commercial solutions. Those needs include the cost of creating, maintaining, and supporting tests as well as integrating the testing infrastructure into the broader development and release process and culture.

Assembling a complete solution from piecemeal tools requires significant time and effort, especially when incorporating open source tools that lack professional support and service level agreements. Those costs pale in comparison to the cost of maintaining and improving that solution over time and at scale. Moreover, companies will be responsible for ensuring that test writers have the resources they need to be productive.

These costs are very hard to anticipate and estimate, especially when multiple components with differing update schedules and compatibility goals are included. For large, mission-critical or long-lived projects, this rabbit hole could be the difference between eventual success and failure.

Commercial tools can mitigate that risk by offering built-out capabilities, documentation, roadmaps, and well-defined support arrangements. Teams can verify that the core offering satisfies their technical needs, integrates well with their internal infrastructure, and that it will be stable under heavy loads, before investing heavily to create a custom solution.

Choosing or designing a testing solution boils down to understanding the technical needs and capabilities of the team, and comparing the cost to build and maintain a custom framework against the cost of licensing a commercial tool that can be leveraged by in-house resources. Teams should consider not only the testing process itself but how well it can be integrated into the automation and reporting process to ensure it is providing clear value.

Conclusion

Comprehensive solutions such as Sencha Test enable JavaScript developers and test automation engineers to perform cross-browser, cross-platform, unit and end-to-end testing. Software development teams can speed up the process of test creation, automation, and execution, and they can improve collaboration among developers and test automation engineers by leveraging a test automation platform that helps teams deliver high-quality, sophisticated web applications to their customers faster.

Sencha Inc.
1700 Seaport Boulevard, Suite 120
Redwood City, CA 94063 USA

U.S. Toll Free: +1 (800) 212-2802
International Sales: +1 (650) 299-9550

www.sencha.com
Email: info@sencha.com