A Survey of
Multi-Browser Testing Tools

Michael Hodnett
Sandi Tepper
December 15, 2014

Provaré Technology, Inc.
4555 Mansell Road, Suite 210
Alpharetta, GA 30022

Email: info@Provare.com **www.Provare.com**

Table of Contents

1. Introduction

Testing web applications for "cross-browser" compatibility is well understood to be a painstaking, arduous task, and it affects software development teams of all sizes. When one considers number of combinations possible among the browser platforms and operating environments of any given web user, the obvious reason it is so troublesome is easy to see. However, an even better understanding is obtained when the compounding effects that software variability and change can have on both browser platforms & operating environments during the development & testing process are taken into account. Web-applications developers who need full compatibility can easily attest to this experience directly. Perhaps this is part of the reason why some web applications are developed and tested on only **one** browser platform, and later validated towards the end of development in other browser platforms (if at all).

As supportive technologies across the digital spectrum evolve, it is getting even more difficult for development teams to keep up with and choose the right testing methodologies for cross-browser compatibility. A very common practice is to wait until customers complain. At that point, a developer can go look for the reported issue in a specific browser and fix it.

Another strategy is to learn and apply testing tools during the implementation of functional requirements. However, development teams may should carefully consider the relevance that differing browser-platforms and operating-environment versions have on their functional requirements before development starts. And then they are faced with the daunting task of choosing the right tool in enough time to install, learn, and configure the tool to perform adequate testing without destroying their timelines. Between these two extremes lies the realm of manual cross-browser testing and tools that promise to simplify cross-browser testing.

The "battle of the browsers" is undoubtedly fierce and dates back to the Netscape days. We have recently begun to experience a similar phenomenon with operating environments. This is significant because of the ever growing ubiquity of mobile devices, as well as the up-and-coming internet of things. Both desktop and mobile browsers and operating environments (on any device) can target the same end user and must provide a functionally equivalent experience. Let's consider the following thought experiment to see how:

> Examine the preferences of a targeted user. This particular person may access a web application based on a variety of hardware devices in their possession. She may own a desktop PC running Windows 7 on which IE11 and Firefox are installed. She may also own an older Android-based tablet but uses a new iPhone as her primary on-the-go mobile device. If this person resides with a significant other, he might access the same web application on an early 2009 iMac running its version of Safari, an iPad from 2010, and a recently purchased Android phone. When this dynamic duo travels together, they may find themselves in an internet café accessing the web application on Windows 8.1 with IE11, or perhaps the latest iMac with Chrome.

Taking all these factors together reveals, in key ways, the stifling reality that web-application developers have been managing over the years as they try to guarantee accessibility to a fully feature-rich product across a diverse ecosystem of content delivery. But with the continuous evolution and growth in technologies, there is much more to manage now than there ever was in the days of Netscape. Only in the last 5 years have responsive templates become mainstream, effectively placing themselves as a new and critical concern for developers who need to provide high quality user experiences. Over the next five years, technologies will evolve further, and as a result, will confound web developers' ability to guarantee compatibility in all relevant environments. This is why it is even more critical than ever for development teams to have a solid testing strategy so they can maintain momentum during the development & delivery of fully feature-rich web application experiences.

As Quality and Productivity Specialists, Provaré is constantly on the look-out for new techniques or tools that can help our clients to be more effective and efficient in their development processes. Long gone are the days when "Just test it in whatever browser you have" is the direction from the client (or the recommendation from us, for that matter). In order for companies to be confident that their products are suitable for all of their target users, regardless of the access platform, testing simply **must** be done in multiple browsers running on multiple operating systems across several hardware form factors.

2. The Problem – Cross-Browser Compatibility Testing

Cross-Compatibility Testing is not about pixel-perfect replications across browsers, but about equivalent *functionality* across browsers. The stark reality is that there are so many browser platforms, operating environments, and combinations that taking them ***all*** into account while simultaneously implementing functional requirements can quite literally be an impossible task.

Most web-application development teams will know some of the decisions that their targeted web users have made in the digital supply chain as to at least platform (desktop or mobile), OS and browser. There are generally analytics available to development teams to identify their main audience's preferences. Once this is known, the business can make a decision on the supported platform / OS / browser / version combinations.

Even with the "what we will support" decision having been made, there could be 10 or more of these combinations which must provide a functionally equivalent experience for the end-user. In order to make sure that happens, testing across all of those platforms must be performed to validate each combination works as intended by both the developer and the business.

The simplest approach is also the most expensive: repeat every manual test on every combination. You can then be assured that every unit of functionality works in every one of the supported systems. In the case of 10 supported combinations, however, you have now increased your QA costs by 10x. While there are certain applications for which the cost of a defect is high enough to justify that expense, the vast majority of applications would be crushed under such a cost burden.

An approach that is gathering interest in theory is the use of tools that provide cross-browser testing, thus "automating" the process and theoretically saving 9x of the 10x in cost. This

sounds like the perfect solution for the folks who are only interested in the color of the ink on the project. Unfortunately, they very rarely find that the cost-benefit ratio is that skewed in reality. There are many tools available and, as one would expect, they all have their pros and cons. What none of these tools are at the moment is a complete replacement for manual testing across the platform/OS/browser combinations by an experienced, qualified Test Engineer.

The following sections in this document detail some of our research into this area, focusing on tools that can operate behind a firewall or through a VPN since that is our most common environment with our customers. As we look for ways to be more efficient with our clients' timelines and budgets, we are investigating tools that can facilitate that effort. Some of these tools have merit but, again, none of them are yet at a stage of maturity where we would be willing to stake the Provare reputation on the reliability of any of them.

During this evaluation, we did not compare the ramp-up time and script development time required to create the automated tests with the time required for manual testing. Obviously, this would be a critical factor for any company who is looking to replace manual testing with automated testing. None of the tools we evaluated appeared to have sufficient promise to warrant moving on to this step at this time. We are hopeful, given the progress in these tools over the past few years, that they will reach a point where such an analysis is warranted and useful in the not-too-distant future.

3. A Discussion on Tools

We initially considered 26 tools, with a mix of past mainstream usage, current mainstream usage, non-mainstream tools, and even tools that showed some nonzero promise in testing web applications. Of this set of 26, 4 appear to have been retired. Surprisingly, a couple of those were well-developed tools supported by very big names like Adobe. Such big names currently recommend other tools for testing web applications.

Out of the 20+ targeted tools that remained from our initial selection, not all tools allow testing to be conducted in the same way. In fact, a number of tools do not allow testing in multiple different browsers (i.e., Firefox, Chrome, IE, etc.); rather, they allow testing across multiple versions of a given browser (like Firefox).

Other tools support testing through the implementation of browser engines.[1] Some such tools lock you into a specific versions of a browser engine. Some merely scan websites and create a collection of static images that may be manually inspected later. So while they save some manual clicks, they are still heavily reliant on manual inspection and on the inspector's diligence in spotting layout issues.

[1] Also called "layout engines" or "rendering engines," browser engines actually render html or xml into the layout that you see in the browser user interface and are also used in other applications that require such rendering capabilities such as email programs web development tools, etc.

Some of the most popular tools currently in use are cloud services utilizing virtual machines. There is also a hybrid approach of cloud-based services coupled with a desktop agent for testing the cross-compatibility of web applications.

3.1. Testing Across a Firewall

Cross-browser testing across a firewall is currently a limiting factor on a large number of cloud based services. However, some of the top contenders in this market do provide tools or mechanisms to support tunneling to a protected web server.

The tools covered in this report include the following:

1) BrowserStack
2) SauceLabs
3) Spoon.net
4) Lunascape.tv
5) SmartBear's TestComplete
6) AutomationAnywhere.com
7) BrowseEmAll
8) GhostLabs

Each tool has a number of strengths and weaknesses with respect to simple tests. The main focus of our investigation was on the ability of a tool to first support useful cross-browser testing and then to support testing behind a firewall. The evaluation progression for each tool was on setup, then ease of learning, and finally the testing itself.

It is worthy to note that two tools are full automation tools that claim to support cross-browser testing (i.e. AutomationAnywhere & TestComplete), while all other tools are targeted more narrowly at a subset of Cross-Browser Testing.

3.2. The Deciding Factor for Each Browser

Each of the tools listed above were included in our list based first on their prevalence of use in the main-stream by big players and second, based on their promise as up-and-coming tools. Then the ability for each tool to support testing behind a firewall was critical. The actual ability to perform useful testing across a firewall was established by verifying that each subject tool could access one of our client's development sites. Emulation of the cross-browser tests and a determination of our comfort-level with the results was the final piece of the puzzle. Pricing models were collected to complete the picture.

4. Desktop Only VPN Solutions

4.1. Automation Anywhere (AA)

"Automation Anywhere is an intelligent automation software for IT and business processes. … Automation Anywhere features … [a] web recorder, variable and debugging support, task-

chaining, conditional, file, system, database and Internet actions, single click web data extraction, Turbo-speed and IE plug-in." [1][i]

As a full automation tool, AA was chosen for its claimed ability to support testing in multiple browsers. AA is a record – modify – play model as far as cross-browser testing is concerned. The script is recorded by the tester; the recording is modified to be compatible with another browser; the modified script is "played" in the second browser. AA has features to support creating automated scripts with respect to uniqueness of browsers (e.g. network latency). Other tools (like TestComplete) are better designed for such tasks; however, AA's capabilities in this area are still very strong.

AA is limited to recording scenarios in IE only which must be installed on the test machine. According to AA's LiveHelp, these scripts can be played back in other browsers. However, creating these scripts is not straightforward and the effectiveness still needs to be determined.

Since AA is a UI automation tool (i.e., is not limited to browsers per se), you can literally record any action or event occurring in the Operating System during a general recording. This implies that you can open other browsers and record user's actions in those browsers. However, the general recording feature (i.e., the feature that allows recording of any action in a Windows environment, whether in a browser or not) doesn't account for network latency when retrieving a website in these browsers and network latency could render a general recording useless. For example, if one were using the general recording as one navigated around the website, a delay on the opening of a page or tab could end up placing a mouse-click in empty space on playback. From that point the recording becomes useless. The *web* recording feature (again, only in IE) ***is*** capable of handling this latency.

AA is a very powerful automation tool and for being a fully featured tool, it has one of the lowest learning curves. For strictly cross-browser testing, the inability to record in browsers other than IE and the time required to develop useful play-back robots for other browsers limits the usefulness for replacing manual cross-browser testing.

4.2. TestComplete

TestComplete from Smart Bear also provides automation outside the scope of multiple browser testing. Smart Bear is making a very strong name for their entire set of tools and TestComplete delivers, but not without some issues with respect to cross-browser testing.

TestComplete is also a record – modify – play model as far as cross-browser testing is concerned. If the "automated" test needs to be transferable to other machines, there are dependencies in the way TestComplete records that may limit its usefulness. Browser windows are best recorded while maximized in order to promote transferability between machines. However, this implies a potential issue with screen resolutions, since larger resolutions on the playback machine can make the browser's rendering show up in parts of the screen that were not part of the original recording.

Another larger issue occurred when using Chrome for recording an application built using Flash/Flex. The PPAPI Shockwave Flash plug-in (a very common plug-in for most browsers)

was not supported by TestComplete during our testing. The following unresolved issues were present:

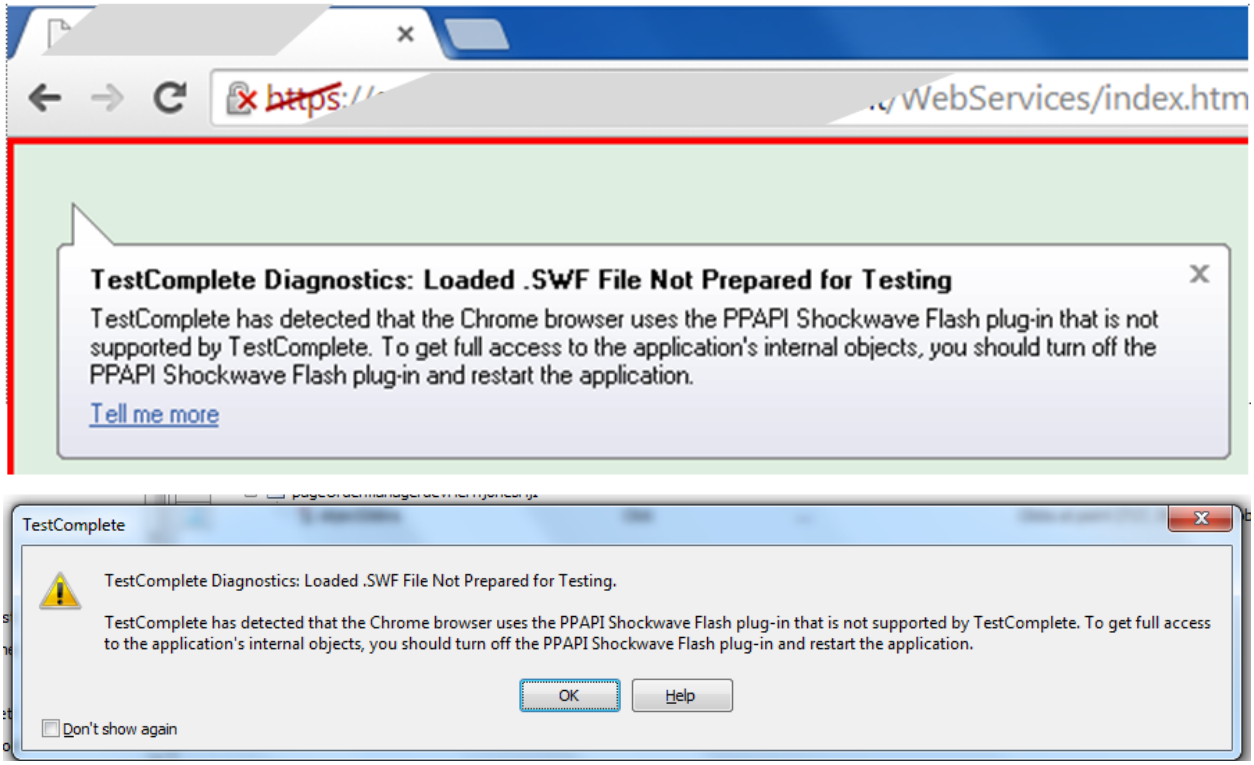

Figure 1. TestComplete Errors on Flash

This tool is a bit more challenging to learn compared to AutomationAnywhere's tool. However, with enough time and familiarization, productivity in either tool would likely be the very close. If Flash is not used in the application being tested, TestComplete could be a good adjunct to a manual test suite.[2]

4.3. Vanamco's GhostLab

GhostLab is the first tool to allow parallel testing across multiple devices and browsers (rather than record-modify-play) in a very intuitive way. The idea is that GhostLab acts as a proxy server AND a web server. So, when GhostLab is running on one machine, other machines on the network connect to the GhostLab server via a dedicated IP address using a local browser and the tests conducted on the host machine are executed on the connected machines as well. Client browsers require no special configuration or modification. The goal of connecting a device and browser to the GhostLab server is so that all devices' browsers can be tested by a single tester.

[2] While the death of Flash has been predicted for a while now, it seems to be hanging on quite well. At some point the HTML5 revolution may occur and this will no longer be a limiting factor.

GhostLab works behind a firewall for the machine hosting the GhostLab server. This allows multiple browsers on the same OS to be tested in parallel. However, connecting *multiple* devices to the GhostLab server over a VPN failed.

Two experiments were conducted to test multi-device access over the VPN. One was to connect a Virtual Machine to the VPN using the *same* credentials as the GhostLab host PC (two devices with a common user on the network) and the second was to use different credentials (two devices with different users on the network). In this configuration, neither of the users was able to find the GhostLab server IP because this client's VPN security forbids access to any local machines on your internal network. In order for GhostLab to work with the application on the opposite side of the firewall, the security settings must allow access to your local network at the same time as the remote network. For some networks, this may be allowed. For our customers, this is usually not the case.

In addition to the problem with the VPN security, there were other troubling results during our testing with GhostLab. For example, it was possible to invoke scrolling in one browser without it registering in another. During another test using an iMac as the server, the screen on the iMac showed that it was stuck in the middle of submitting a form but in fact the form was submitted.

In terms of learning to use the tool, GhostLab was hands-down the easiest in this group.

With enough screen real estate to show the connected devices, GhostLab is an excellent choice for manually testing a script once in parallel across 3 or 4+ browsers. The Test Engineer must be careful, though, to be clear with their client that some issues may be flagged using GhostLab that would not have been if the browser were tested natively (see scrolling issue above).

There are caveats to this recommendation which make GhostLab a poorer choice:

1) If you need to test multiple browsers on a *single* device
2) If you do not have enough physical devices on which to install one browser on each to test the range of combinations desired

The Test Engineer may still be more efficient by combining even 2 or 3 devices at a time as opposed to testing each individually as long as they take the proper precautions against false reports.

4.4. Lunascape.tv

Lunascape is designed to run three different browser engines: IE, Chrome, and Firefox. This tool works just like you would expect a browser to work. One downside is that the user is limited to specific browser versions which may or may not be useful to a client's need. The much larger downside is that this tool does not allow the user to test in parallel across the browsers. With this being the case, it seems to have very little additional value beyond being free tool and having a quick learning cycle.

Lunascape might be very useful when you need to stack different browsers on-top of each other so you can compare the differences in rendering for certain HTML structural elements

and you don't want to install the actual browsers on the machine. For example, you may need to make sure the element containing an image is appearing in the correct spot of the screen across all browsers. Except for the fonts, one is quickly able to see how the structure of the site is rendered across the different engines, and can take corrective action if it is slightly off.

About that "except for the fonts" statement…font rendering is different for each engine, which is somewhat expected. However, there were differences in the font rendering of a similar engine used between Lunascape and the actual browser. In our tests we used Lunascape v6.9.1 which implements IE's Trident Engine (v. 10.0.9200.17068). We compared the font rendering to that of an actual IE browser with a very slightly different engine (v. 10.0.9200.17089). One would expect the fonts to be rendered the same way, but this was not the case. In our testing, the font differences were noticeable but not significant enough to cause structural misalignments. In other applications this might be more significant.

Lastly, a key question for us when testing Lunascape was reliability. There were some random bugs that happened consistently in a couple of the engines. One example would have caused a false defect to be filed using Chrome's WebKit in Lunascape. The following image shows a page appears to not have loaded anything but this was not reproducible during native testing of the actual browser.

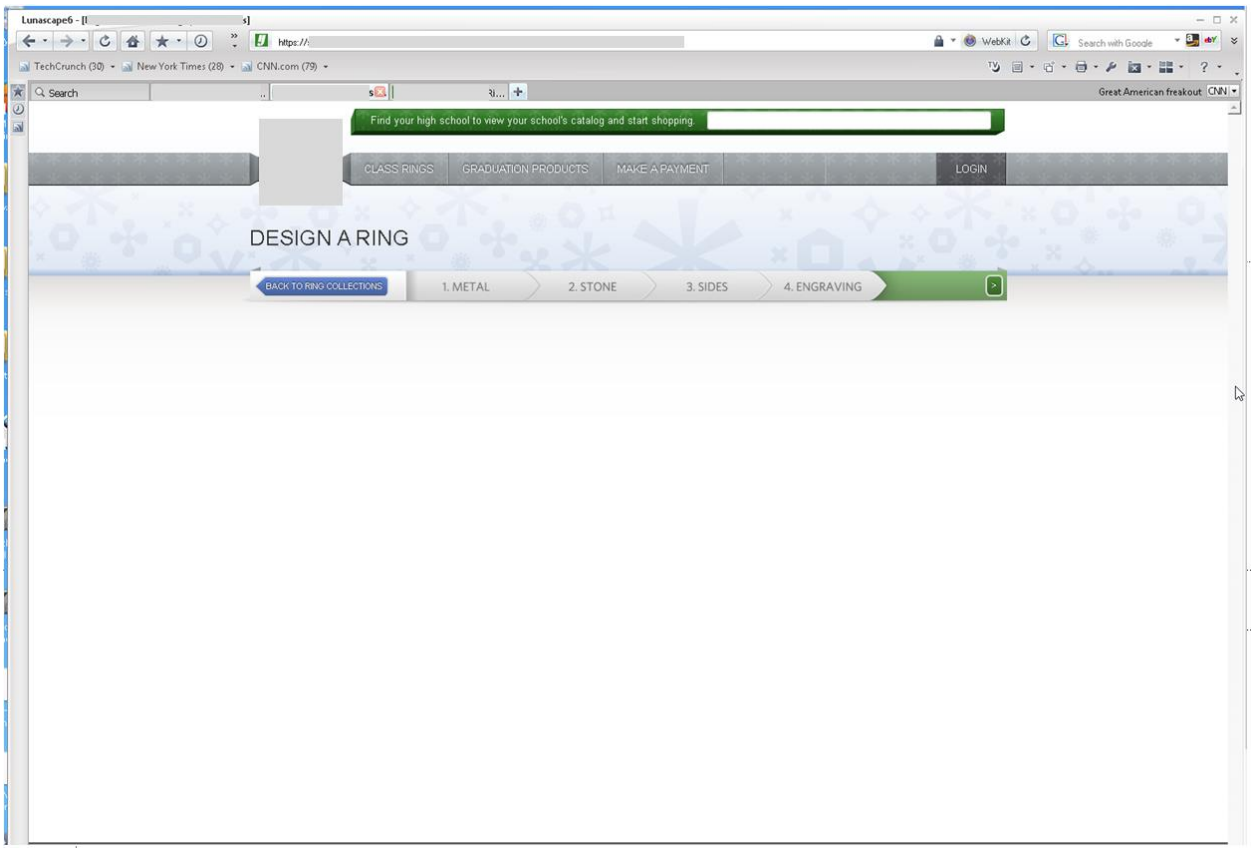

Figure 2. Lunascape False Defect

4.5. BrowseEmAll

BrowseEmAll seemed at first to be the most reasonable tool for multi-engine support cross-browser compatibility testing. It offers most if not all of the currently major browsers and versions that are likely to be in use. It is also relatively easy to learn and is one of only two tools to provide parallel testing in more than one browser. This alone that makes it light-years beyond Lunascape.tv.

However, this tool appears to be suffering from stability issues. When loading a client site using a Chrome engine the tool crashes every time. We did not do in-depth troubleshooting on this as the other engines appear to be able to load the same site without problem. See the figures below for a comparison of BrowseEmAll's Chrome and IE10 engines.

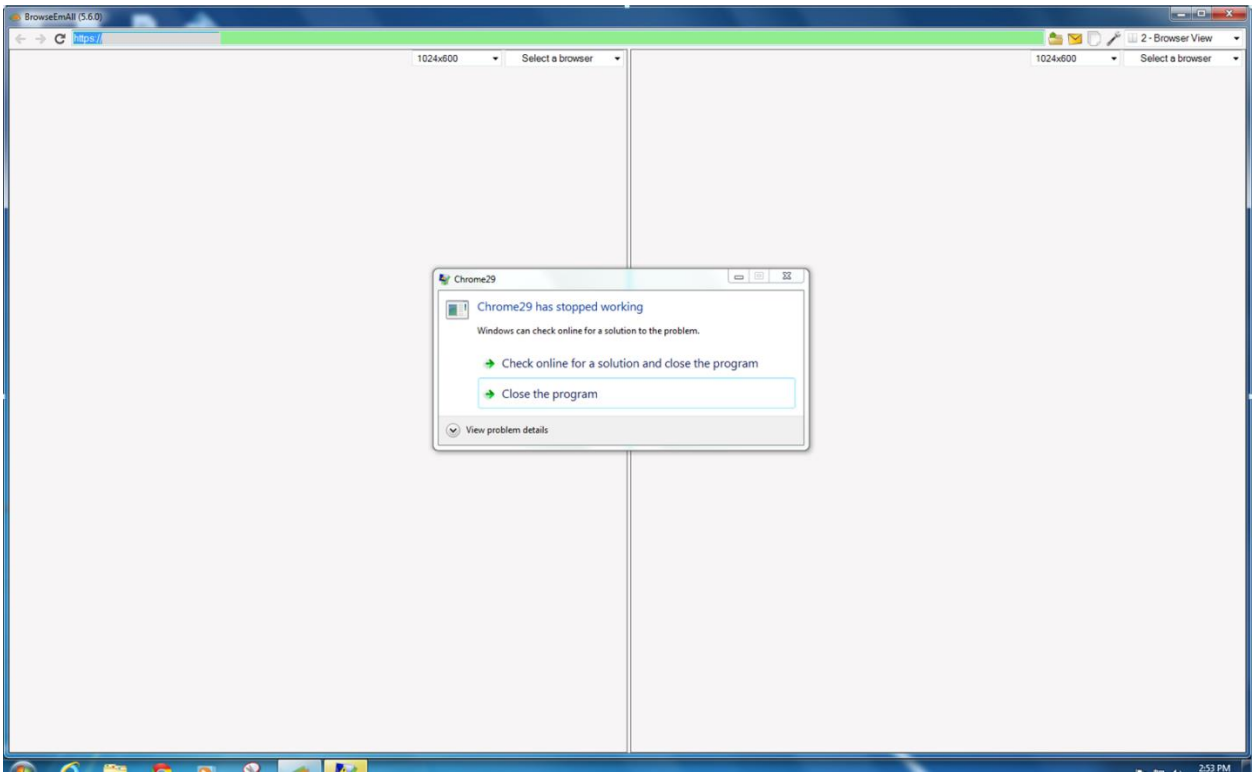

Figure 3. BrowseEmAll Chrome Crash

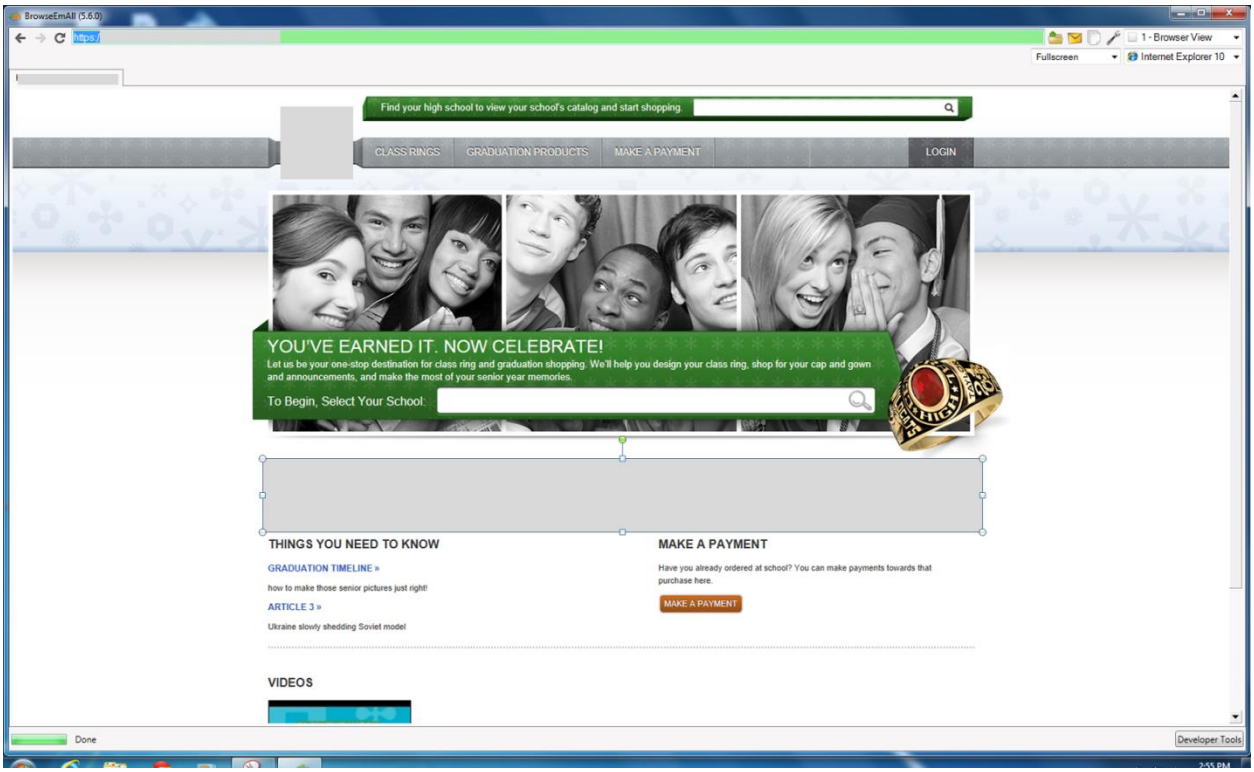

Figure 4. BrowseEmAll IE10 Engine

Another issue is rendering “smears”. When using the parallel feature, scrolling is synchronized across the browsers but it appears that rendering degrades. Notice in the close up that the “Your Name (required)” label is partially covered by the smear.

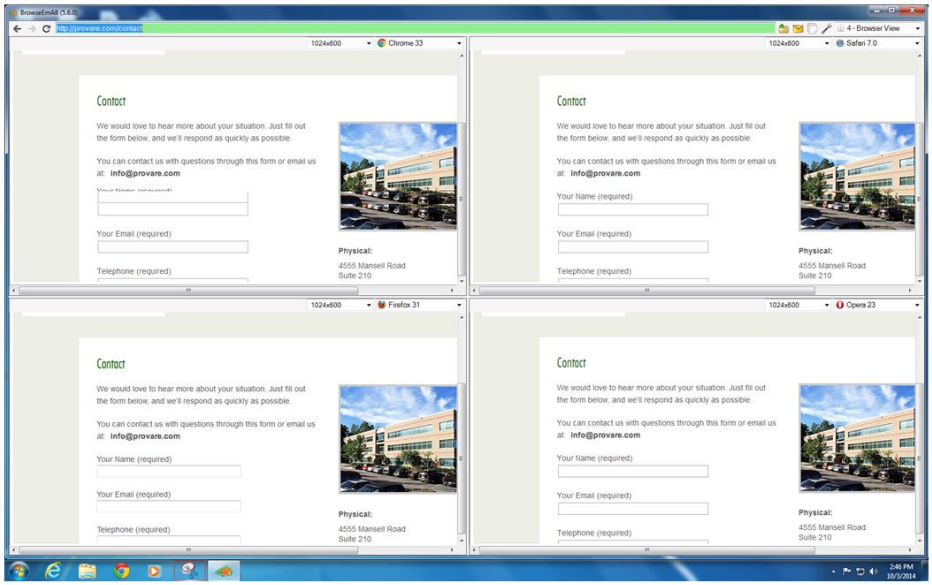

Figure 5. BrowseEmAll Parallel Testing

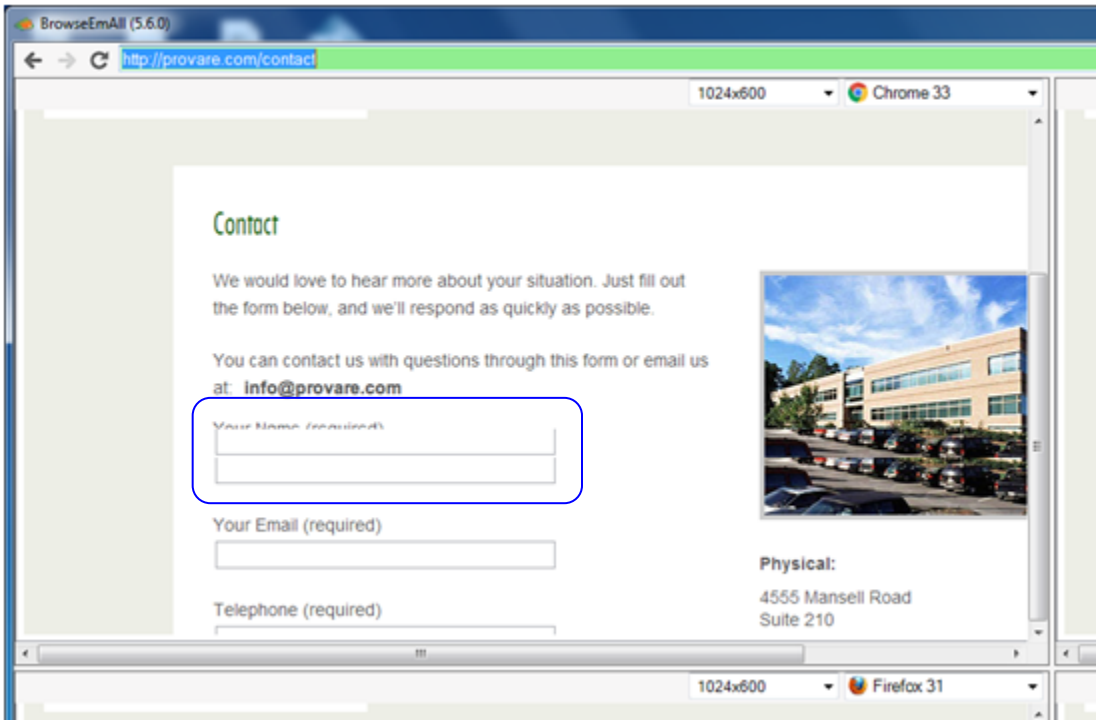

Figure 6. BrowseEmAll Smear During Scrolling

Overall, BrowseEmAll is a paid service with potential stability issues that could interfere with testing efforts. The degree of interference would need to be evaluated for each testing project to know whether it would be significant enough to warrant using a different tool.

5. A Hybrid Cloud & Desktop Solution

5.1. Spoon.net

Spoon is a tool that offers a desktop application that appears to act as its own virtual environment in which real versions of pre-configured browsers can run on a host operating system, but they are run in a way that doesn't make the browser become part of the host operating system. Spoon does require some browser pre-configurations and actions to be taken through the Spoon's SaaS interface prior to testing.

Spoon's Browser Building Studio is executed via the Web and allows you to select one browser out of the three major players: Chrome, FX, and IE. You then configure your browser's environment to include Flash, Java, Silverlight, etc. You can also select browser extensions based on what is available for the chosen browser. Once your browser is configured, you can launch it through the site onto the Spoon.net Desktop Application. This application must be installed beforehand.

During our testing, the fastest way to begin was using only a browser with no other environment configuration (unless absolutely required) since Spoon downloads and prepares each environment on the fly. The more configuration options you choose (e.g. Java, Flash, etc.) the longer it takes Spoon to download and prepare the environment for you to run through the desktop Spoon-App.

The user interface is not entirely intuitive but should be relatively easy to understand for experienced Test Engineers. There are not that many features to learn and understand thus making the learning curve for this tool relatively small. However, the user must be familiar enough to know which plug-ins are needed in order to avoid getting "stuck" because Flash isn't installed, or something similar.

One very confusing problem we had was that an Internet Explorer 6 environment was set up in one instance (Figure 7) but then we received an error message stating only IE8+ was supported in another (Figure 8).

In any case, Spoon.net does not allow the user to test multiple combinations at the same time. It is only a tool that allows access to browser environments that the Test Engineer might not have installed on their test machine. Spoon.net does not alleviate the 10x cost for manually testing the 10 combinations from our original example.

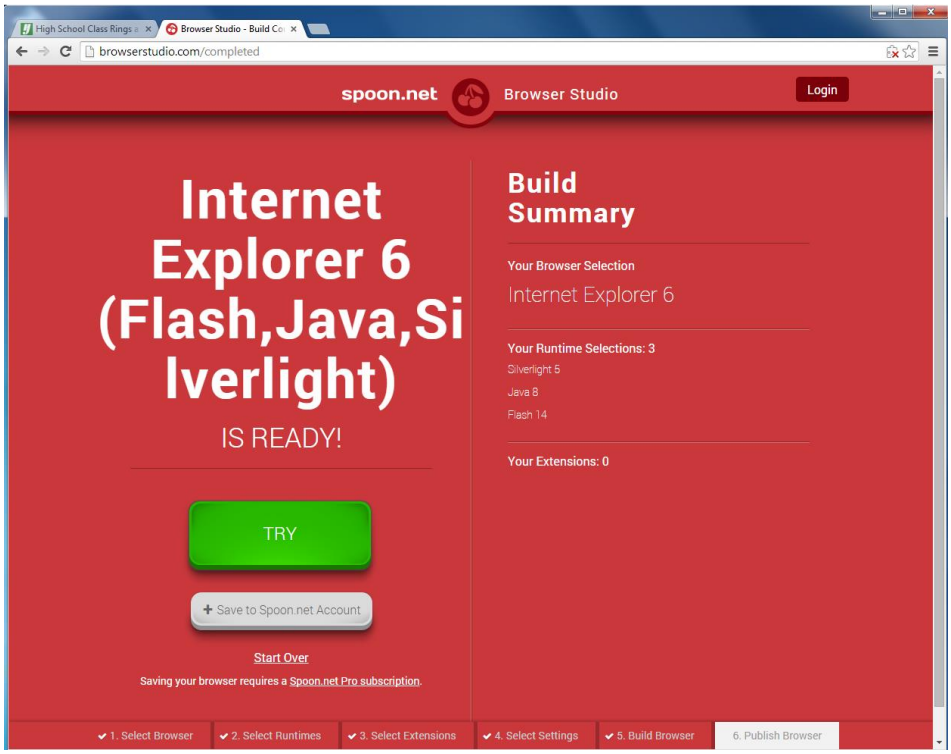

Figure 7. Spoon.net Pre-configured Environment Built

Figure 8. Spoon.net IE6 Not Supported

6. Cloud-Only VPN Solutions

When cloud-based solutions are seriously being considered, BrowserStack and SauceLab will be the go-to sites. Their pricing models and how they handle firewall access differs. BrowserStack came out on top between the two in a number of significant ways in our testing. But SauceLabs has been around longer and is a relevant force in VM based Cross-Browser Testing.

6.1. BrowserStack

BrowserStack is a cloud service offering VM-based deployable testing environments in which real browsers can be tested. It has integrated SSL ability which allowed testing of an internal site quickly, easily, and without a glitch. The learning of cloud based services in general is fast since one only has to learn a specialized tool and BrowserStack was no different.

Like most VM-based cloud solutions, the pricing scheme makes running multiple tests concurrently by one user (note, this does ***not*** mean parallel testing) a challenge, but one that can be easily overcome.

> "Concurrency" in this case means a tester would be executing a single test back-and-forth between two different VM cloud environments so the tester takes *the same physical action twice*. If concurrency happens fast enough, it gives the illusion of parallelism but does not actually reduce the number of actions by the tester.
>
> "Parallel" testing means that the tester executes a physical action only once in only one browser and then that action is executed across all other browsers by the tool. When you have 3 testers working at the same time, this is another form of parallelism, although testers may execute actions at different rates.

The idea of concurrency is important because the BrowserStack pricing model only allows logins from one BrowserStack-Account per browser session. This "browser session" has nothing to do with the browsers we want to test inside the VMs of the Cloud Service Provider. In order for a tester to be able to conduct the same test on two *different* VMs in the cloud, the tester needs a way to create two different browser sessions in order to access two different BrowserStack-Accounts according to the Terms of Service. For example, suppose you open Chrome and log into the first BrowserStack-Account. Now, with that first account and first browser session, you can start conducting tests on the first environment. For the same tester to work across another VM in the cloud, they have to start another, independent browser session. In Chrome, this can be done by using Incognito mode. This is an effective way to test two VMs at once, but Chrome doesn't support adding a third session. At this point, you would need to start a different browser all together to access BrowserStack. This technique can be used to effectively allow one user to access up to 4 or 5 VMs at once in the cloud.

We admit that the explanation above seems complex, but the learning curve is still relatively light, and it depends on a hosting environment (where the tester will work) having a sufficient number of browsers to access the required number of VMs.

Even if the number of browser sessions can be achieved, the tester is still manually executing all of the steps in each of the browsers. Again, the 10x cost increase in time / effort is not alleviated here. Only the access to the particular systems needed for testing could be solved using BrowserStack.

6.2. Sauce Labs

Sauce Labs works in much the same way as BrowserStack does. However, Sauce Lab differs in that it has extra steps necessary to enable a secure tunnel for use in a VPN. The configuration includes installing a third party ssl agent (SauceConnect). The agent must then be running before beginning tests in the cloud.

7. Findings

While we have discussed the variety among the tools quite a bit, we want to also share a very important factor that emerges when one examines these tools for testing web applications: any tool used for cross-browser testing at this point must first be tested itself in order to ensure that it will accurately reflect what would have been rendered by the browser being emulated. In our research, we did not find a single tool that didn't, at one or more points, either indicate false failures or false passes of important tests.

False failures are fairly easy and inexpensive to deal with… If a failure is indicated, the tester simply logs it and goes back later and re-tests it in the native browser.

But false passes are much more problematic. The expectation ***is*** that tests will pass. Verifying each passed test is equivalent to having run the entire test suite a second time. If a tool *ever* correctly renders objects when the native browser would have failed to do so, then the entire engine emulation of that tool is suspect and cannot be relied upon.

Additionally (and ironically) none of the tools that we tested were *themselves* bug-free. In our analysis of these tools, we found one or more bugs in all of them, many of which were serious impediments to the tool's setup or use.

So which tool is right for you? The more complex web applications become (i.e. ones that are feature-rich and dynamically driven), the more critical it becomes that you will need to not only choose a tool for testing web applications, but you will need to choose a tool that has been very well tested and can confidently be considered reliable during the testing process. At this point, for our purposes, "very well tested" would have to mean that ***we*** have tested it. Our experience during this analysis would prevent us from trusting the QA efforts of the tools' publishers.

At this time, we do not have a recommendation for a tool that can solve the problem of cross-browser testing when a VPN or firewall is employed. None of the tools we evaluated were able to significantly eliminate the duplication of actions by the test engineer without significant development effort. Given that each development must then itself be tested and maintained as software, none of the tools were seen as viable options to manual cross-browser testing.

Advances in the tool development world will likely make these conclusions moot in the next few years and we are looking forward to revising our conclusions.

8. Additional Recommendations

Some readers may be wondering "Just how do you do cross-browser testing in any sort of cost-effective way?" That's a fair question. At Provaré, we use a sparse-matrix concept that allows for all functionality to be tested more than once while spreading testing across multiple browser, OS, and platforms (form factors) including mobile devices. We find that this works to give our clients the reliability they need across various platforms without a ridiculous increase in cost. If you need testing of your application or product across multiple operating systems, devices, and/or browsers contact us to see how we can help you achieve this in a cost-effective way.

[i] CNET.com, "Automation Anywhere". Retrieved: October 2014 Access: http://download.cnet.com/Automation-Anywhere/3000-2084_4-10346863.html