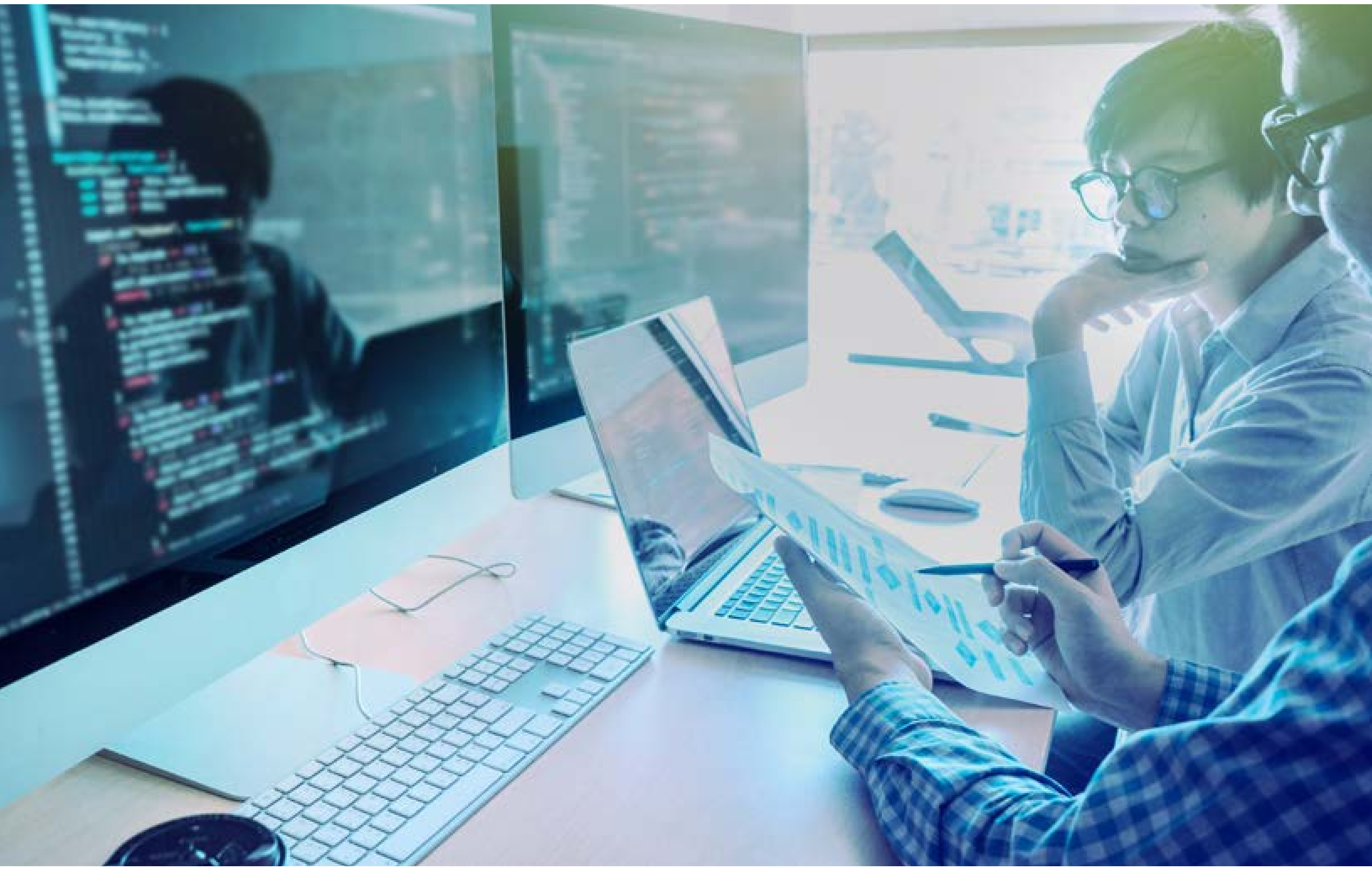
PROD PERFECT

WHITE PAPER



The Seismic Shift in How We Test Software

Dan Widing, Founder & CEO

TABLE OF CONTENTS

Introduction 3

The Rise and Fall and Rise Again of Browser Automation 4

Recent Innovations in the Space 6

What it Means for Software Testing Today 8

The Next 5-10 Years 9

About ProdPerfect 10

INTRODUCTION

As it's been since ARPANET, functional web software today is mostly shipped by luck, duct tape, or sheer will. Ask any engineer with tenure at an ecommerce company and they can tell you about the last time they broke checkout, the defining feature of doing commerce online.

Every year we have groundbreaking technologies that change the entire game of what kinds of software we develop: virtualization, the cloud, mobile, SOA, REST, NOSQL, ML/AI, microservices, SPAs, serverless, the list goes on. But except for the very bleeding edge of talent and investment, software has been tested pretty much the same way over the last 20 years: a mix of human manual testing, and a shallow layer of test automation.

As late as 2018, even sophisticated businesses struggle with testing:42% of software companies are still testing entirely manually; only 32% have become mostly automated, according to recent research sponsored by testing companies Sauce Lab and SmartBear. 75% of testing teams aren't keeping up with software development: code is being deployed only partially tested. The majority of testing teams don't even have a firm practice for test management—they aren't even certain what they're testing. The bugs that are not caught by testing cost the globe literally trillions per year. This is true despite the fact that 25% of software budgets are allocated towards testing for QA.

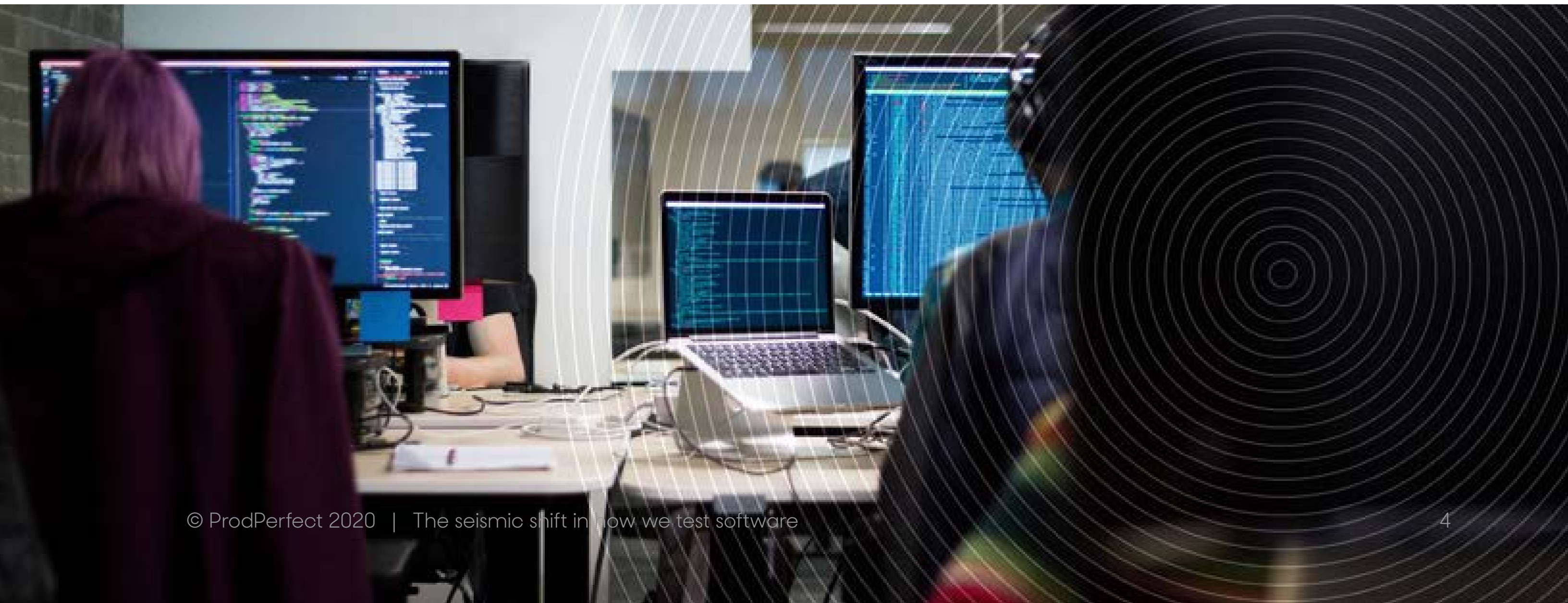
We continually hope to be building better software year over year, but the bugs have been inescapable. Humans are imperfect and inconsistent which, while beautiful in our way, you'd never build a human pyramid more than a couple of layers high, less you invite catastrophe. We're slower to the task than machines, we behave inconsistently, and communication breaks down at sufficient scale or with multiple layers of the organization. With any manually tested system having bugs, project failures and high development costs are the expectation and the norm.

However, there's a light at the end of this tunnel. The last 2 years have seen a new breed of tools appear that have the chance to change the game. We ourselves roll our eyes when products throw the words machine learning or artificial intelligence at their own product. But suffice it to say the tools are getting smarter in a non-trivial way.

THE RISE AND FALL AND RISE AGAIN OF BROWSER AUTOMATION

As an industry, we've tried and failed to get away from using a browser to test web applications. We tried unit testing, we've tried API testing, we've tried behavior-driven testing, we've tried test-driven development, and we've tried just monitoring. None of these are necessarily bad, but it's pretty rare that they provide a truly conclusive test that the product you are shipping will work for its users. Every one of those approaches tests a limited understanding of the application and there will always be gaps in the testing that will let significant bugs get through. Good teams have as many nets as possible so that each testing system's blind spots are hopefully covered by something else, but it's impossible to have certainty. And the worst blind spots are usually where the integrations and units come together. At the end of the day, a web application is exercised by users through a browser: the only way to really know if it will work is by testing as much as possible through a browser preferably as close as possible to how real users exercise the system.

For all the advancements in other tooling and successor frameworks over the last 15 years, the standard in Software Quality Assurance Engineering is Selenium. Originally developed in 2004 the idea was simple: rather than ask a human to manually click through a browser to test features of a web application, you could write code that would do this work. Originally a framework to remote control a browser it has evolved to be an abstraction for controlling or deeply integrating into a wide variety of browser-like systems as well as large scale collections of remote machines that could run the browser.



While the initial principle is simple—automate a browser—the results can be complex. Selenium's evolution has spawned incredible layers of abstraction to handle all the different varieties of browser behavior and usage that one could dream of, as well as where either the browser can be run or and abstractions to fit how the developers want to write their tests. The dirty secret is that even were it a simple browser, it really isn't just testing one unified component. Automated browser tests are almost always testing an independently standing environment. This means there's a standing network setup, server setup, database, and external services for each test system involved. Since each of those components then have both a constantly evolving state and processes for being updated, there are many independent variables that need to be controlled. The most frustrating impact is on time-based behavior. Loading a web page kicks off a variety of synchronous and asynchronous processes which makes deciding when to call a page fully loaded and ready to test tricky, particularly with the rise of single page applications. Mash all these frustrations together and you get modern Test Engineering.

As such, maintenance burden and stability issues have continued to plague browser testing. Test suites have continued to be maintained manually, by humans updating code, across changes to the underlying application. Stability issues and complexity have resulted in tests that are flaky, they sometimes pass and sometimes fail for the same version of the application. These problems have meant that browser testing has remained expensive, frustrating, slow, and ultimately of limited effectiveness. In 2015, Google was officially advocating severely limiting the number of browser tests a software team should maintain. The whole process left much to be desired.

Various other tools emerged to improve the efficacy of browser testing: various record-and-play tools and other "scriptless testing" products were designed to allow less technical and less expensive resources to drive automated testing-these tend to be particularly unstable. Products such as BrowserStack and SauceLabs made cross-browser testing much more accessible, instead of needing to maintain a series of remote machines hosting various versions of various browsers you could instead pay a service to do that for you.

Browser testing has had an ongoing less-acknowledged but still serious problem in its entire history: ultimately, software teams have to guess what's important to test. No data drives the decision of what to test. Software teams get together and decide what they believe are the most important use cases on their application which require testing, and go test those. A gap always exists between what developers predict their users will do, and what they actually do. This means that critical bugs which affect core user flows can be missed, even when a test suite is well-maintained. It also means that many tests cover irrelevant or rare use cases, so browser suites become even bigger, and thus slower, and harder and more expensive to maintain.

"Browser testing will achieve its full potential when it runs faster, costs less to build and maintain, and (most importantly) runs consistently, throwing alerts only when it is catching bugs."

RECENT INNOVATIONS IN THE SPACE

Browser testing will achieve its full potential when it runs faster, costs less to build and maintain, and (most importantly) runs consistently, throwing alerts only when it is catching bugs. A number of recent innovations in the space have brought browser testing a long way.

Crowdtesting

Crowdtesting is a process by which the software team provides test cases to the vendor, and the vendor provides a large bank of people to manually perform the scenarios. It has a few advantages: it's easier to set up than your own automation suite, it requires less ongoing maintenance than a home-built suite, and manual testers can sometimes catch bugs that automated tests would miss. However, this approach has several drawbacks. There are a few major players in this space.

Because customers pay for each test run, more software shipped correlates directly to more money spent. While manual testers can sometimes catch bugs that automated tests would miss, they will also frequently report false positives or miss other critical bugs, due to the inexactness of a manual review by an untrained/unfamiliar resource. In addition, while the only real maintenance is updating test instructions, it still means that a resource has to be assigned to the task, continually updating and changing the test cases to prevent the test from becoming stale and outdated.

Crowdtesting is much like the American military: it was an innovation to win the last war, not the next one. The machines will provide far more resource-efficient, consistent, and easy-to-use testing products that will leave Crowdtesting as a footnote that ultimately served as a stopgap between the old way and the new way of testing.

"Autodetection tooling analyzes user traffic to determine test cases that represent common flows of user behavior, and then Autogeneration automatically produces repeatable test scripts based on those test cases."

Machine Learning (ML)-Enabled Record-and-Play

With Machine Learning (ML)-Enabled Record-and-Play, a third-party application adds an additional layer to your own application, allowing you to build tests recording you using your software. These tests are intended to be functional through many small changes of your software, by building "models" of the test event, rather than using conventional testing hooks. This reduces test maintenance costs and significantly reduces instability. Because the tests are truly automated (rather than crowdsourced), you don't have to pay for the cost of running each test. There are a few big players in this space and perhaps a few dozen others.

However, since it is your team developing the tests with the external application, the gap between your team's understanding of the application and actual user behavior remains. Additionally, the tests need to be rebuilt every time there's an appreciable change to the product, requiring attention and input from a software team. Lastly, since tests all run through the interface, if you decide to leave the service, you take no assets with you–you're back at square one.

Ultimately, we believe the core problems of browser testing won't get solved until machines can help with deciding what to test, in addition to helping with how to test an application. Ultimately, good browser testing exists to make sure users can do what they want to on an application, without encountering bugs. If you're able to test what users are doing, you'll make sure they don't see bugs.

Autodetection/Autogeneration

Autodetection/Autogeneration is where machines begin to help to decide what to test. Autodetection tooling analyzes user traffic to determine test cases that represent common flows of user behavior, and then Autogeneration automatically produces repeatable test scripts based on those test cases. This process can be repeated continuously to update a testing suite. The players in this space have emerged more recently and are fewer in number.

The last challenge for Autodetection-driven technologies is anticipating changes to User Interfaces (UIs). Ideally, browser tests would only break when a true bug has emerged. When UIs change dramatically, even machine-driven tests will fail when the change is deployed to a testing environment. In a short time, it's likely that these technologies will be capable of detecting and prioritizing developer behavior in pre-production environments to automatically update tests to anticipate these UI changes.

WHAT IT MEANS FOR SOFTWARE TESTING TODAY

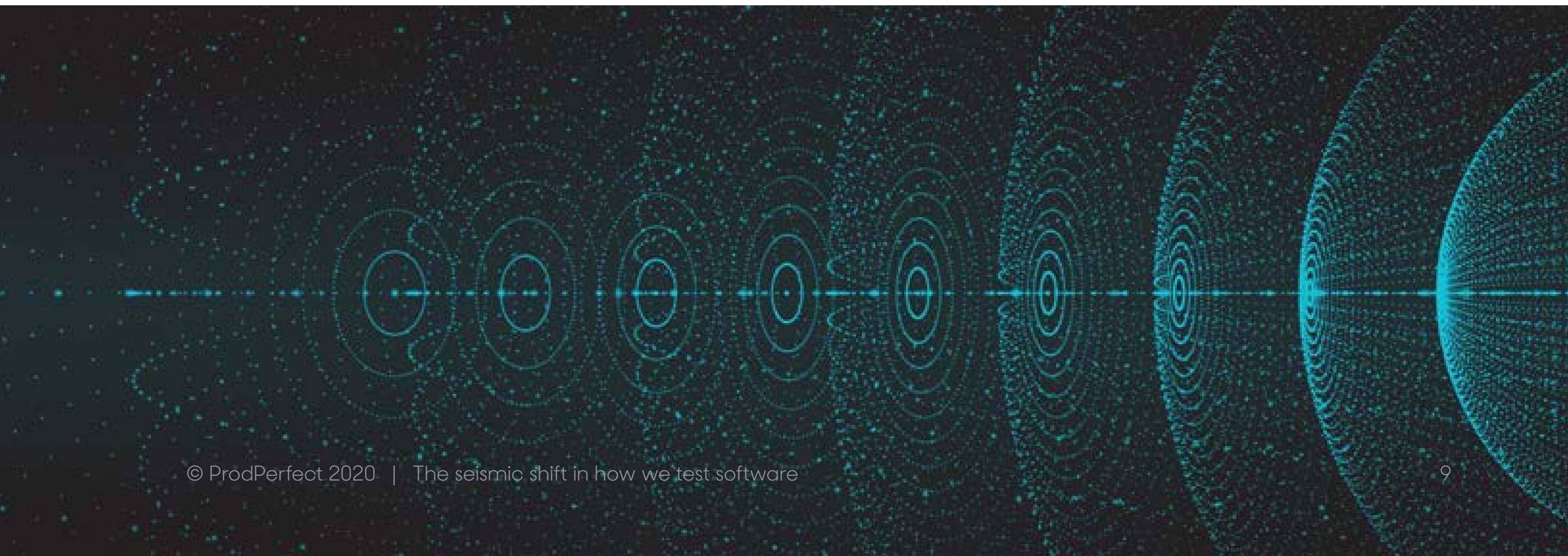
Improvements in data science and data engineering (behind true Machine Learning and most "machine learning" tools masquerading as true ML) have unlocked quite a deal of potential in helping reduce the cost and instability of browser tests. Machines are able to do more of the work. If we think of the history of browser testing over the past 25 years, the path has been something like this:

- **Humans decided what to test, and humans tested it manually**
- **Humans decided what to test, and humans wrote and maintained code to have machines do the testing**
- **Humans decided what to test, and machines helped low-cost humans more quickly do the testing**
- **Humans decided what to test, and humans told smarter machines how to test with less babysitting and maintenance**
- **Machines decide what to test, and machines do the testing**

Steps 4 and 5 represent the recent seismic shift in testing. We're now seeing an emergence of a few technologies that will cut the human out of the loop completely, both in deciding what to test and translating those tests to machines. The result will be test suites that catch more bugs, update and run faster, and require no human effort to build or maintain. We suspect that in 5 years, machines will own the entire browser testing process.

THE NEXT 5-10 YEARS

The processes that we're seeing today are just the beginning. What's happening is that we're unleashing data to improve the way we test the web. First, we were collecting data between executions of each of our test runs to improve our expectations. Now we're updating our expectations of what should be tested based upon the behavior of the user. These expectations and test behaviors will only get more intelligent over time as we're ultimately simulating users. These user simulations will eventually get sufficiently accurate and intelligent that we can expect to rely on it as the primary form of web software testing. The goal is for the tools to recommend potential solutions to the problems they have seen in the past.



ABOUT PRODPERFECT

Unleashing the power of machine learning to solve the hardest, most important, and previously unsolved problems in end-to-end (E2E) QA testing, ProdPerfect is the only autonomous E2E regression testing solution on the market that continuously identifies, creates, maintains, and evolves E2E test suites via data-driven, machine-led analysis of anonymous live user traffic. It is a fully-managed testing solution that addresses insufficient test coverage which causes critical and costly bugs in production; removes the burden that consumes massive engineering resources; and eliminates long test suite runtimes that slow deployments and decrease developer velocity.

SCHEDULE A PRODUCT INTRODUCTION TODAY

PRODPERFECT