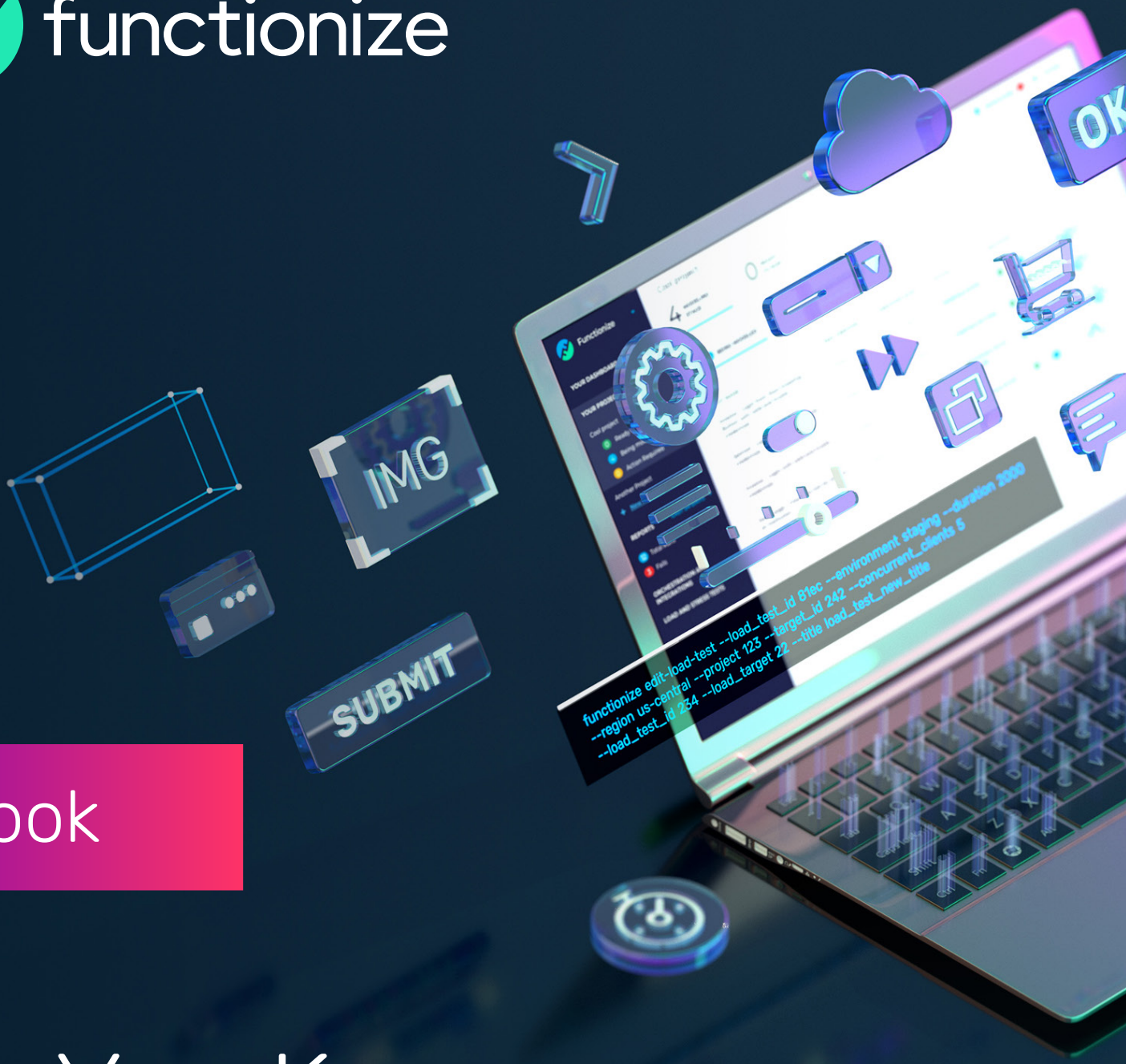
functionize



eBook

Do You Know The **True Cost** of **Legacy Testing?**

How to migrate painlessly

Autonomous testing is steadily replacing test automation as the default for UI and web app verification. However, migrating from your old system can seem like an impossible step. Many teams feel they are trapped in a migration Catch 22. They fear that migration will take too much time and effort, disrupting their delivery cycle. But the longer they leave it, the worse this situation becomes. In this whitepaper, we want to show you how easy it is to migrate to Functionize, as well as highlighting the true costs of sticking with your legacy system.

The argument against change

In any business, change can be painful and hard to achieve. This is especially true when talking about changes to your product delivery pipeline. Some businesses rely on getting new releases out of the door to stay alive and many need it in order to continue growing their customer base.

If you introduce a new testing approach, your team will need to learn how to use it properly, integrate it into your delivery pipeline, and then migrate all your existing tests to the new system. All this while trying not to disrupt the ongoing testing and delivery of your product. As a result, there can be apparently compelling arguments against changing.

> It'll cost too much time and effort.
>
> It will disrupt delivery of our new features.
>
> How will we know it's working correctly?

However, we want to show you that such arguments are specious. To do so, we will draw on the experiences of real customers.

The issues with legacy test systems

Test automation was revolutionary back in the day. However, test automation has been stuck in a rut for many years. Test creation often still relies on handcrafted test scripts. Test analysis is seldom automated. Worst of all, the brittle nature of most test automation tools means test maintenance dominates a QE's workload. Let's explore each of these in more detail.

Test creation

Test automation tools rely on scripts to select and interact with the system under test. These scripts are often written using domain-specific languages, such as Selenese. Even where they use JavaScript or Python, writing tests require real skill. Sadly, the number of Developers in Test with sufficient skills is pretty low. To try and get around this, various test recorders have been created. The best-established of these is Selenium IDE. This plugin for Firefox allows you to record the interactions you made with the system under test. These are then used to generate a basic test script which can then be adapted and tailored. More modern recorders seek to be a bit cleverer than Selenium IDE. However, the resulting scripts often need to be refined by hand.

Another issue that dogs test automation is that these tools were designed in a simpler age. Nowadays, websites often use deeply nested DOMs, meaning many elements simply aren't selectable by the test automation system. At best the test system knows there is an element present, but actually accessing it can be nigh-impossible. Responsive sites mean that the HTML is generated on the fly to suit the device it is being displayed on. This takes the pain of cross-browser testing and multiplies it many times over.

Test analysis

Once upon a time, analyzing test results was relatively easy. You would know in advance the result of any given interaction with the system under test. This meant quality engineers became used to using direct comparators to check if the test was successful or not. The problem is, modern websites and web apps are usually completely dynamic. Content changes depending on the user, the time of day, the device being used to view it, and even the geographic location of the user. In short, life has become much more complex for test automation engineers.

Nowadays, you have to construct far richer comparators to check for the correct outcome. For instance, in a shopping site, you may need to check whether there is a valid price, in a

valid currency, and with the correct amount of sales tax added. Ultimately, it often requires potential test failures to be manually verified, at least until you have sufficiently refined the script.

Test maintenance

Before the days of test automation, no one had heard of the concept of test maintenance. Nowadays, many test automation engineers might wish they still had never heard of it! Test maintenance is the process of updating your existing tests to cope with changes to your system. The problem with most legacy test systems is that they are extremely brittle. A simple change to the UI or layout can break some or all of your tests. This is because even apparently simple changes can mean that the test script no longer knows which selector it needs to choose. As a result, you will see a large number of tests failing.

The only solution is for your test automation engineers to manually check every test failure and work out if it was genuine or just a maintenance issue. If it is the latter, they need to update the script and rerun the test. Unfortunately, it can be hard to tell the difference. This is because the actual trigger for the failure often happened several steps earlier in the test. Take an eCommerce site with two “buy” buttons on the page. The first is for the selected product, the second is for a special offer. If the site is redesigned so these buttons move, the test script may select the wrong button. But this is still a valid action, so it won’t break anything. It’s only when you reach the step where the test verified the cart contents that you see an error.

The true cost of not migrating

The combination of the above factors has a dramatic impact on the productivity of your test team. Have you ever looked in detail at what they are doing? Do you know how long it takes them to create a new test? How efficient are they at analyzing test failures? Most of all, have you asked them just how much of their time is wasted in maintaining existing tests? In our experience, test creation using legacy systems is often a matter of hours or even days. Analyzing failures can depend to a good degree on how complex your system is. But the biggest issue is test maintenance, which often occupies 50% or more of their time. And this is entirely wasted resource! Imagine if this was happening in any other part of your business. Imagine if every time your developers made a small change you had to refactor your codebase. Or had to redesign your production system I’m sure you’d be looking to fix it pretty quickly! But somehow, it has become the accepted norm in testing.

Solving the issues with legacy test systems

Here at Functionize, we have set out to make testing truly autonomous. As a result, migrating to Functionize is pretty painless. Let's see how our technology helps you with migration.

Adaptive Language Processing

ALP™ is our proprietary natural language test creation engine. Creating new tests is as simple as writing the steps down in plain English. ALP™ then takes these steps (aka test plan) and converts them into a validated test. It does this using NLP (natural language processing) and other forms of AI. The resulting test runs on the Functionize Test Cloud.

ALP™ understands keywords like "verify", which makes it easy to construct self-checking tests (e.g. ones that make sure the correct page has loaded before trying to interact). Best of all, it can directly ingest the output from most test management systems. As a result, it makes migration a doddle. All you need to migrate your existing tests is to feed your existing test plans into our ALP™ engine.

Fingerprinting and Self-Healing

Functionize's autonomous testing system is based on a number of machine learning approaches. When you specify an element in a test, this element is fingerprinted and analyzed. It records hundreds of data points for every action including API calls, element visibility, and server responses. This fingerprinting also means that our system copes perfectly with nested DOMs. Because the element is being selected intelligently, it doesn't matter if it isn't in the parent DOM, unlike with legacy test automation systems.

As each successful test run completes, the ML model updates the system's understanding of what every element does. This means that simple UI changes and layout changes are no problem for the system. Just redesigning a button doesn't change how it interacts with the system. Equally, moving the button on the page doesn't alter its behavior. A manual tester would know this, and so does our system. Put simply, it Self-Heals. This ability to cope with elements that move means that our system is particularly suitable for testing responsive sites.

Adaptive Event Analysis

Our AEA™ engine lies at the heart of our autonomous testing. It is designed to dramatically reduce the burden of test maintenance.

Root Cause Analysis uses a combination of a rule-based expert system and machine learning in order to identify the most likely root cause of a given failure. This advanced modeling system relies on historical test data and an understanding of common failure scenarios in order to help identify the root cause of the current test failure.

Having identified the cause of the failure, you might have a large number of possible resolutions. The **Smart Suggestion** engine uses historical data and machine learning in order to determine what the correct action is most likely to have been. Using this, it is able to automatically make a number of suggestions for possible failure resolutions. These suggestions are presented to you. Having chosen the correct resolution, the test is updated and re-run.

One Click Updates is the highest level in the AEA™ hierarchy. Often, there is more than one potential root cause for a given failure, and each of those root causes may have several possible resolutions. In situations where the correct solution may not be immediately obvious, the self-heal feature allows the test to be re-run with each of the potential solutions. The results of these are then ranked and presented to the user who can then click to approve the best resolution. This will automatically update the test for future runs. The idea here is to replicate what a user would have to do if they were manually updating the test. This feature has the potential to save a vast amount of QA maintenance time.

What benefits have our customers seen?

6x increase in productivity across your whole team

90% cut in test creation time through using ALP™

80% reduction in time needed to diagnose test failures

90% less time spent on test maintenance.

We make it a point to ensure that customers are happy with the results of migrating to Functionize. And in most cases, they couldn't be happier! Here are a few quotes from happy customers:

Using Functionize I was able to create new tests in minutes instead of hours.

Wilson, TOTVS Labs lead QA Engineer.

We needed a platform that was going to enable this level of automation … and you guys checked all the boxes.

Kyle Sylvestro, CEO of Sytrue.

With Functionize, I spend almost no time manually maintaining our test suites.

Leslie, SyTrue non-technical employee turned test engineer, thanks to Functionize.

One day we had restyled a page pretty drastically. The head of QA came to me and said she couldn't believe it. None of the tests had failed. All the effort the QA team puts into writing tests is never lost. The QA team can spend more of their time on creating new tests and less on maintaining brittle old tests.

Todd Erickson, Director of Technology, Agvance.

Before Functionize, our Quality Engineers had to rely on our DevOps and back-end engineers to perform their work. Now with Functionize, they can not only serve more engineers and cover more parts of the product, but they can also perform their job without the help of other teams.

Vicente Goetten, Executive Director, TOTVS labs.

1-800-826-5051
functionize.com

156 2nd Street
San Francisco, CA 94105