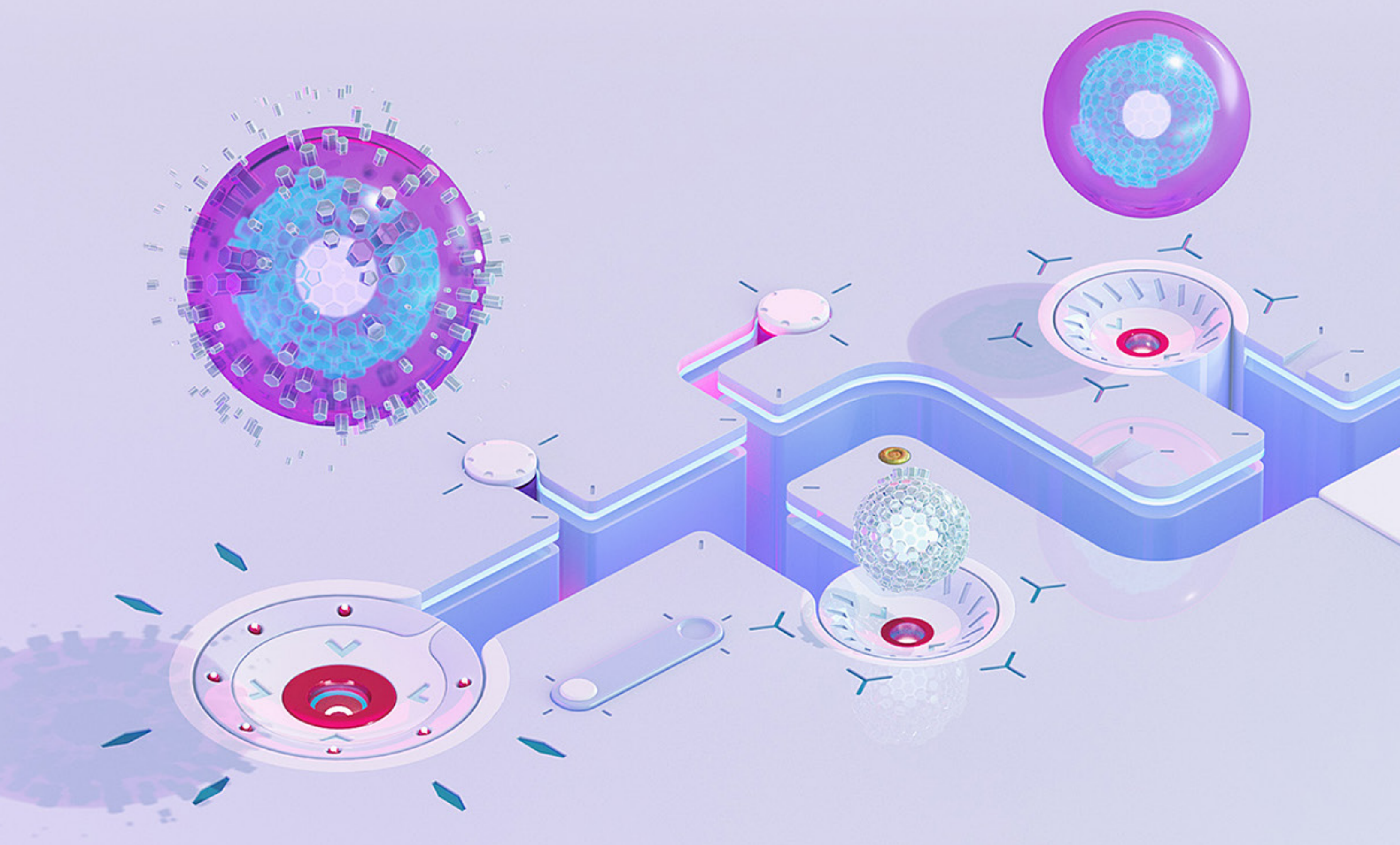
functionize

Why test automation needs machine learning

"Software testing has endured, what I term, a QA Winter. Developers and testers still maintain tests the same way they did in the early ages of the internet. Test automation has fallen far behind — and at Functionize we are on a mission to change that.

Tamas Cser, CEO of Functionize.



Executive Summary

Functionize was born out of the observation that test automation tools were failing to keep pace with the advances in the applications being tested. We have set out to revolutionize testing by applying the latest advances in machine learning and data science. As we will explain, machine learning can drastically boost how test automation can get your products to market faster without falling victim to brittle tests more likely to break than the code itself.

The issue with test automation

Generally, test automation suffers from issues at every stage in the testing lifecycle.

- **Test creation** is often slow and requires skilled Developers, even for the most basic tests.
- **Test maintenance** eats time dealing with false-positive failures if you update your UI.
- **Debugging** is slow and especially painful to do when testing cross-browser and cross-platform.
- **Test execution** is often slow and limited by the test infrastructure, often requiring a Devops Engineer to be heavily involved for any kind of scale.
- **Analysis** only checks a small part of the overall UI, so it can miss key bugs and defects.

Many of these challenges prevent companies from having the elusive Continuous Deployments many strive to achieve for the fastest time to market. Even when investing in a team to create and maintain a test suite, often these only cover the simplest tests and harder ones will perpetually be *done later.*

Data Science and Machine Learning as a solution?

Over recent years, it has become abundantly apparent that data is power. Many newer companies have turned to analyzing this data to try to solve various problems using machine learning (ML) For test automation, some key applications of ML are:

1. **NLP or Natural language processing** - teaching computers to understand human language. This offers the potential of computers that understand pre existing manual test plans.
2. **Computer vision** - computers identify and categorize objects within images. This can help with the analysis of test results to see when the application under test has changed.

3 **Anomaly detection** - the system spots unexpected outcomes from a test and can help with maintenance, debugging, and analysis.

We will explore these in more detail in the rest of this eBook. We will also answer some key questions: Why do you need test automation in the first place? What is machine learning? And how does ML help with test automation?

The need for test automation

Before we dive into ML and its application in testing, let's start with a review of a few foundational concepts. Testing is the process of validating an application to ensure it does what it is supposed to do, and does it well. Ultimately, this is meant to identify bugs, defects, or bad user experience (UX) before it's in the hands of your users. While bug

and defect are often used synonymously, a bug typically refers to when the app does something it wasn't meant to do. This unexpected behavior may result in the app freezing or even crashing, but there are also more subtle bugs, like the app giving you the wrong result. A defect, however, refers to when the app doesn't meet the original specification. Defects are most often seen in the user interface (UI). Lastly, bad UX typically means the app slows down or becomes unresponsive, or the UI doesn't behave as the user expected.

There is a huge variety of tests that you need to run on any application, ranging from unit tests for each function through to integration tests to make sure all the complex components of your system under test work together seamlessly These tests can be done manually, but it is virtually impossible to do that without slowing releases down drastically - or just introducing large amounts of risk into the deployment of a release. As a result, it is generally best practice to automate as much of your testing as possible. Unit tests that developers create alongside their code, assuming they are doing so, are quick to write, and quick to run. There are numerous frameworks out there to help you in any language, and many continuous integration tools to help kick off your tests when building your code. However, as you move up the testing hierarchy to functional tests, integration tests, performance tests, and more, tests get exponentially harder to run and harder to automate though these tests tend to be the most valuable to automate.

Functional UI testing is particularly important for modern applications. Since it is the primary way the user interacts with the application, it is also where most bugs tend to reveal themselves. Moreover, users that have a poor experience with an application their first time tend to not try it again. Functional testing takes significant time and resources as every new feature has to be tested thoroughly, and like all new features, can add many new scenarios to test. In addition to testing those new features, you also need to ensure that their addition hasn't broken any other part of the application (regression testing). For a large application, that can easily take weeks to do manually which is why many testing teams put a real emphasis on trying to automate the Functional UI testing.

The issue with UI test automation

The first universal framework for functional UI test automation was Selenium. Since it was created in 2004, Selenium has become the de-facto standard for functional testing within a browser. Selenium allows you to automate complex test cases within a Test Framework such as JUnit or TestNG. However, there are some well-known issues with it that we explain below.

Test creation

Selenium works by interacting with a driver that controls a corresponding web browser. It is able to replicate typical user actions, such as opening a link, clicking on an element, inserting text, or selecting a value in a dropdown. It can also check whether a given element is visible on the page. All these actions are driven by a test script that works with a selenium library. These scripts can be written in any popular language including Python, JavaScript, C#, and Ruby. The problem is, each script is a miniature software project in its own right requiring a test framework, libraries for selenium, and installation of web drivers per browser under test. This means it has to be developed, tested, and maintained. Any tester can learn to write simple test scripts. But to create and manage the framework or complex tests, significant development experience is required in order to make these test frameworks maintainable. This becomes even harder if your application does anything even mildly complex such as using iFrames, a site that lazy loads as you scroll, and more.

Debugging

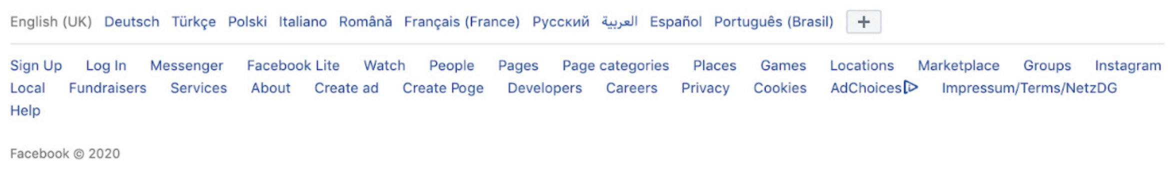
One of the painful things with Selenium is debugging your test scripts. Modern applications are expected to work cross-platform and cross-browser. This means that all your tests also need to work on any combination of browser and platform. Each browser will render a UI slightly differently or will use different libraries, and there are significant differences between each OS. This means you can't simply run your test script on any browser and expect it to work. Typically your test script needs to be rewritten and debugged for each combination of OS and browser. To make things even more complicated, a huge percentage of the population accesses the web via mobile browsers; this requires not just installing a browser on your machine, but installing emulators or connecting to real devices, not to mention the complexity this adds to the test framework. This debugging is almost always manual so creating a single test script can easily take days.

Test Execution

Selenium was created in the days before cloud computing. Consequently, it is designed to run as a single thread application on a server. Selenium Grid allows tests to run across multiple servers, but this is a far cry from cloud scalability. Worse still, all this is happening on your own infrastructure, with all the associated SysAdmin pain that brings. Some companies offer you the chance to run Selenium test scripts in the cloud, but essentially, they are just providing you with a Selenium Grid test infrastructure which they in turn maintain.

Analysis

Obviously, you need to be able to analyze the results of any test you run. Test scripts require you to specify conditions that have to be met for the test to pass. Typically, this is verifying that an element is visible, a field has the correct value, or that you are on the correct page. The problem is, this leaves huge amounts of the UI untested. As an example, can you spot the error in the following modified screenshot from Facebook's login page?



English (UK) Deutsch Türkçe Polski Italiano Română Français (France) Русский العربية Español Português (Brasil) +

Sign Up Log In Messenger Facebook Lite Watch People Pages Page categories Places Games Locations Marketplace Groups Instagram Local Fundraisers Services About Create ad Create Poge Developers Careers Privacy Cookies AdChoices Impressum/Terms/NetzDG Help

Facebook © 2020

Even a human would struggle to spot that it says "Create Poge" instead of "Create Page". And what about fields that change, such as the copyright date which will change each year? Adding explicit checks for each part of the page would take an age. But if you don't do this, you risk missing some key defects.

Test Maintenance

Possibly, the most infamous issue with home grown Selenium is that of test maintenance. Selenium test scripts need to select the correct UI elements to interact with. Typically, this is done using some form of CSS selector, although you can also use other approaches. The test script will simply find the first element on the page that matches the selector it has been given. The problem comes when the UI is updated. All too often, this causes some of the CSS selectors to change. For instance, it might lead to the wrong button being pressed, or text being entered in the wrong field. As a result, many of your tests will now need to be updated. Selectors are also very narrow in their focus, meaning they are only looking at one specific attribute. As a result this may create false negatives whereby the selected attribute didn't change, but another unintended change passes through unnoticed. Reviewing and updating scripts to account for these challenges (often called test maintenance) can eat up 40-50% of your test engineers' time. This is one of the main reasons more mature organizations turn to alternate solutions instead of dedicating so many highly skilled man hours to maintaining custom test scripts.

Background on machine learning

Artificial Intelligence (AI) is transforming many fields, and testing is no exception. A few of these so-called AI systems are actually applying a technique called machine learning (ML).

Machine learning techniques

Machine learning is mostly simply described as creating computer programs that learn to perform a task without being explicitly programmed to do that task. To do so, the system learns to recognize certain patterns and then uses that to trigger actions. There are a few key ways in which ML accomplishes this:

Supervised learning

In supervised learning, you find specific features in your data and assign labels to them. The goal being to teach the system to recognize those features without human assistance after it has been "trained". To train the system, you pass the computer a large set of accurately labeled data. For instance, you might give it a million photos of animals, with every photo of a cat labeled. It then uses these labels to learn to identify cats by creating a model (computer algorithm). You then fine-tune its performance with a smaller set of labeled data in a process known as validation. Finally, you verify that the model is accurate using a small test dataset. Supervised learning is very similar to traditional pedagogic teaching in schools. You show a child an ABC book and teach it to recognize letters and associate the letter with sounds.

Unsupervised learning

In unsupervised learning, you have no existing labeled data. Instead, the model is used to find interesting patterns in the data that might be significant. This is rather like how you might learn a new neighborhood in the absence of Google maps. Unsupervised learning can solve tasks like identifying different segments within your customer base. One of the best-known applications of unsupervised learning is called k-means clustering. This is used to identify clusters within a dataset.

Reinforcement learning

With reinforcement learning, you are helping the computer to learn through a system of rewards. You start with an untrained model. It randomly searches the data and tries to label the features it finds. Each time it marginally changes the model parameters. If it correctly labels a feature, it is given a virtual reward. Over time, the model will become more and more accurate. This is similar to the trial-and-error learning a toddler does when learning to walk.

Composite approaches

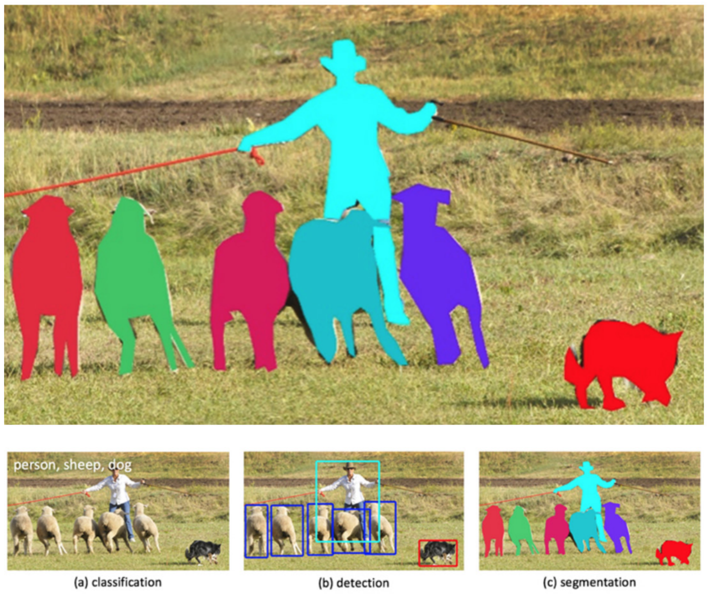
Many practical applications of machine learning involve hybrid approaches. For instance, you might combine unsupervised learning with a small set of labeled features to allow it to quickly converge to an accurate model. Or you can use a technique called boosting to improve the performance of several inaccurate models by combining them. You can also use reinforcement learning to improve the performance of another model. For example, when you are teaching a computer to recognize different accents in speech-to-text applications.

Applications of machine learning

There are a large number of practical applications of machine learning. Most of these are what lay people think of as artificial intelligence. Here, we look at four applications that are especially important.

Computer vision

This is the process of teaching a computer to identify objects within still and moving images. This generally requires several steps. It starts with object localization where the



From https://engineering.fb.com/ml-applications/segmenting-and-refining-images-with-sharpmask/

computer identifies which pixels are related to each other. Next, the computer uses object classification to decide what the object actually is (e.g is it a dog or a cat). Finally, the computer processes the image semantically to understand how the different objects relate to each other. The end result is a system that can identify objects in a picture and recognize how they relate to each other.

NLP

Natural language processing or NLP is actually one of the oldest disciplines in computer science. It involves teaching a computer to understand human language. People have been trying to solve this since Alan Turing first proposed his "imitation game" in the 1950s. However, it is a really difficult problem and was only really solved in the last 20 years. The main issue is that human language is lazy, full of double meaning, and often has a lot of homophones or homonyms. NLP splits sentences into grammatical parts and then reviews how they relate to each other. These parts are then compared against its knowledge of grammar and sentence structure which allows the system to determine the meaning of the sentence.

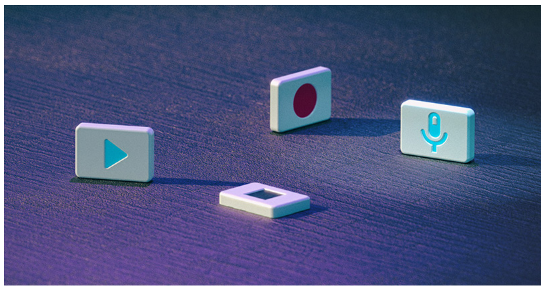
Speech recognition

The prevalence of virtual assistants like Apple's Siri or Amazon's Alexa means we now take speech recognition for granted. What was once the mainstay of science fiction is now in almost every household. Speech recognition requires a computer to combine two things. First, it needs to use speech-to-text to convert the sounds it hears into words. Then it uses NLP to extract the meaning from these. Speech-to-text works by splitting the speech into phonemes. It then uses context and machine learning to determine how to convert the phonemes into text. As anyone with experience with one of these common speech recognition assistants, this is a really hard problem to get right 100% of the time.

Anomaly detection

One of the most powerful applications of machine learning is anomaly detection. That is, identifying outliers within a dataset. This may sound relatively easy when you consider simple 1-dimensional datasets. However, machine learning allows you to spot outliers and anomalies in the most complex of data. Most importantly, it can even spot anomalies in images. This has allowed computers to become as good at diagnosing skin cancers as the best human specialists.

ML-powered testing solutions



A number of companies and open-source projects have worked on solutions to the above problems. But these solutions have met with mixed success. Moreover, none of them address all four of the problems. Let's look at some of these partial solutions.

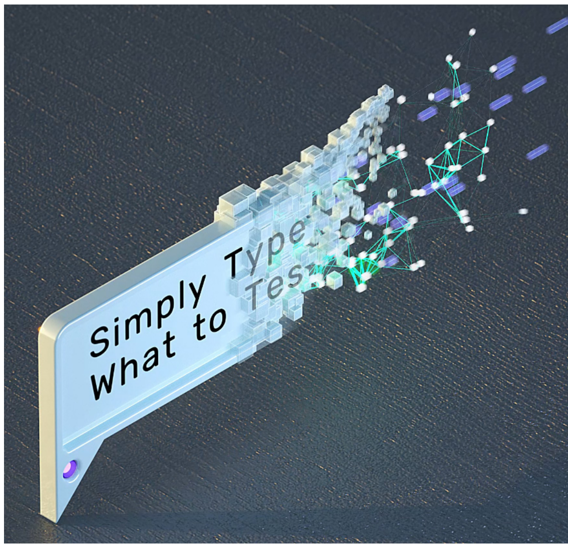
Test recorders

Test recorders are one of the earliest attempts to improve test creation. A test recorder simply records someone interacting with the UI and converts this into a test script. This can then be played back to run the test, replicating the actions initially captured during recording. During the creation process, you can also specify elements to verify on the page, making analysis easier. One of the earliest and best-known recorders is Selenium IDE. While this helps with creating the test script, it can only produce simple tests. Moreover, you still have the same problems with debugging, test execution, and test maintenance. Unfortunately, anyone who has been in the testing industry has seen the evolution of test recorders and often have very negative opinions. Especially after investing time in test creation only to realize the tests were extremely brittle, if they even worked the first time.

Advanced recorders

Fortunately, some companies have created much more advanced recorders in an attempt to address the many shortcomings of traditional test recorders. An advanced recorder might make it easier to add verification steps or branches to your test. This reduces the need for debugging and can make test maintenance quicker. Others also use computer vision (see later) to improve element selection. Though there is currently a broad spectrum of advanced recorders on the market, many offer only slight improvements just generating selenium code. Others are able to do much more than just recording the basic actions and assertions in the test. In order to use all the ML algorithms mentioned above, a significant amount of data is needed; some of these new advanced recorders are able to gather enormous amounts of data in order to solve not just the test creation problem, the all-important brittle execution problem. Much like their more basic predecessors, they still have significant shortcomings. For instance, they often simplify away some complex features, meaning things like data-driven tests are impossible. The resulting scripts are also just as brittle as hand-written ones because they still rely on the same selectors for object recognition at runtime.

NLP solutions



Another recent advance is systems that use natural language processing (NLP) to help write scripts without the need to write code. These systems typically work by using structured natural language. Typically, tests are entered one step at a time. So, a simple test that logs into Facebook might look like this:

4 Navigate to https://facebook.com

5 Enter 'joebloggs@mail.com' in the email field

6 Enter 'MyPassW0rd" in the password field

7 Click Log in

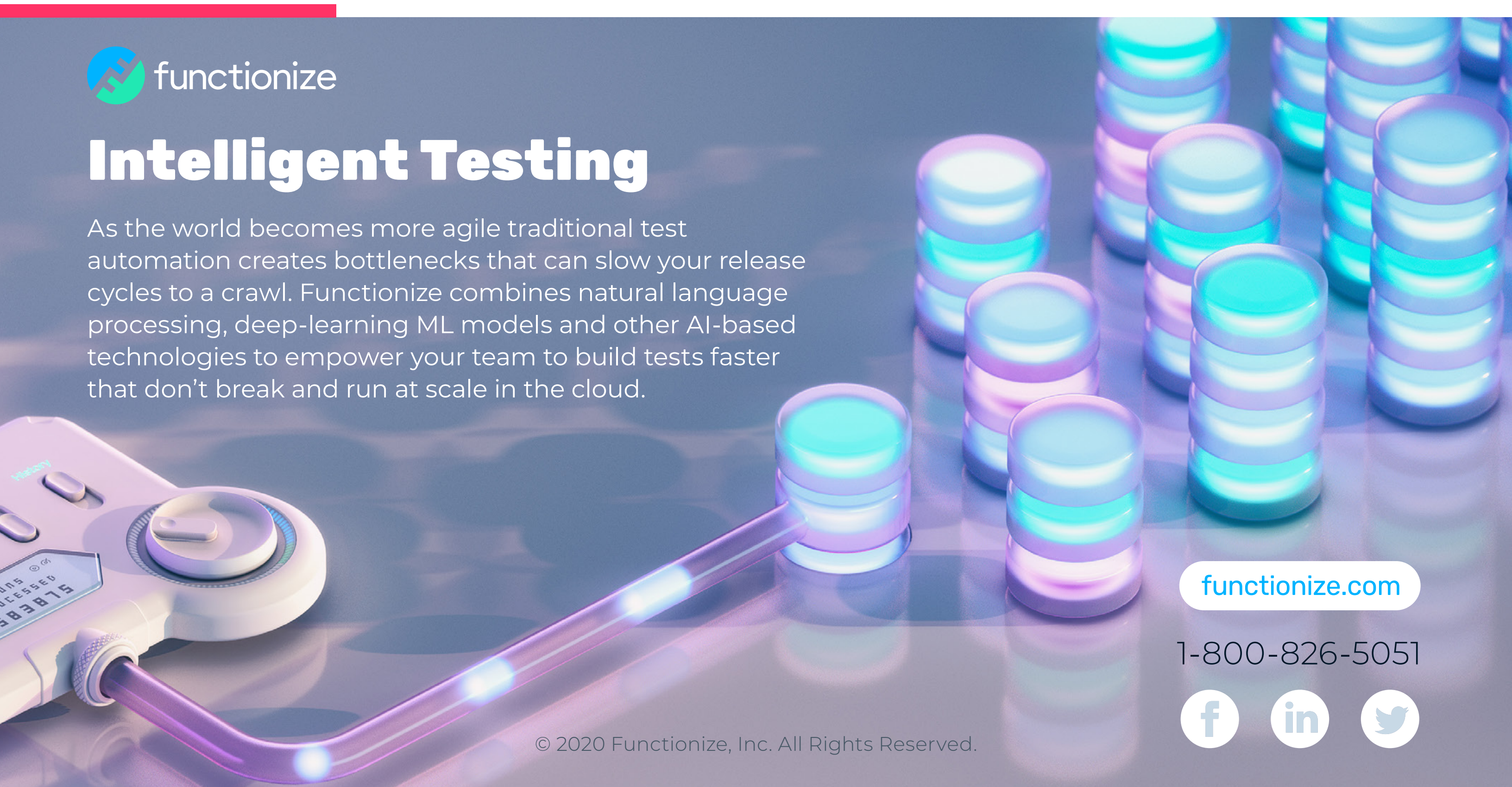
This structured language has similarities to **Gherkin**, the language used in **Behavior Driven Development**. While Gherkin does provide a common syntax across your teams it ultimately is just a layer on top of the normal test creation process, which still requires scripting or using a recorder. By contrast, NLP allows you to use more flexible language constructs and actually creates your test script. This approach simplifies test creation, especially if you combine it with a test recorder, but NLP does struggle with compound test steps or complex validations. It also still faces the same problems with execution, analysis, and maintenance.

functionize

A better way to do things

Functionize is a complete solution for creating, executing, and analyzing tests without the need to write scripts, unless you would like to add advanced customizations or integrations. Over the years, we have embedded machine learning and data science techniques in every part of our product. We have developed a system that simplifies test creation, streamlines debugging, achieves completely scalable execution, and creates durable tests that need little maintenance. The Functionize mission is to bring modern tooling to the testing industry, which has been severely left behind other technology advancements.

In the second part of this series, we explore how machine learning allows our Cognitive ML Engine to deliver truly intelligent test automation. The result is a system that allows you to automate your entire test suite quickly, reliably, and efficiently. Something that is only possible with the intelligent application of machine learning.



functionize

Intelligent Testing

As the world becomes more agile traditional test automation creates bottlenecks that can slow your release cycles to a crawl. Functionize combines natural language processing, deep-learning ML models and other AI-based technologies to empower your team to build tests faster that don't break and run at scale in the cloud.

functionize.com

1-800-826-5051