functionize

Top AI Trends in Testing for 2021

Will 2021 see the end of the QA winter?

Some years ago, our founder, Tamas Cser, stated that the software industry was stuck in a QA winter. Test automation had languished years behind the rest of the software world. Teams were wasting countless hours writing scripts that then needed constant maintenance. Since then, test automation has evolved significantly. AI-powered testing is making leaps and bounds, test automation is becoming mainstream, and even Selenium is evolving. In this piece, we explore how AI and machine learning will continue to drive improvements in test automation. We will also share our predictions for the year ahead.

The changing QA landscape

Testing used to be simple: developers would build a product and pass it to the QA team. They would then test the product and report any bugs to the developers. Once the bugs had been fixed and tested, the product could be released. This model was a great fit for the pre-Agile waterfall world. However, things nowadays look very different.

In the Agile world, the software development lifecycle (SDLC) is much less linear. Now, testing needs to happen in a more flexible manner. At the same time, the products being tested have changed dramatically. We have moved from a world of monolithic applications installed on users' end machines to web applications with cloud-based backends. In some ways, this is a return to the days of central mainframe computers. But today's client machines are infinitely more powerful and capable.

The growth of continuous testing

Nowadays, testing is much more likely to be continuous. That means it happens throughout the lifecycle of a product and includes the entire product ecosystem. The following diagram shows the traditional lifecycle stages for a software product. As you can see, testing is focused in one particular stage, and concentrates on finding bugs in the actual software.

The rise of shift left

Over recent years, there has been a growing tendency for "shift left" testing. That is, to move testing earlier in the SDLC. Of course, some testing has always happened during development. Specifically, unit testing. Over time, this has become more formal, with many project managers insisting on a minimum test coverage before new code can be pushed to master.

Shift left goes further. At the most extreme, you find methodologies like Test-Driven Development and **Behavior-Driven Development**. Here, you start by defining the business requirements for your product. Then you create tests for those requirements. Only then do you start to create the actual code. Everything is driven by the imperative to pass the tests. In turn, the tests are driven by business needs. But many teams are adopting a shift-left approach even without going as far as this. This includes embedding QA team

members alongside developers and running constant smoke tests on the latest builds coming from the CI system.

Shifting into production

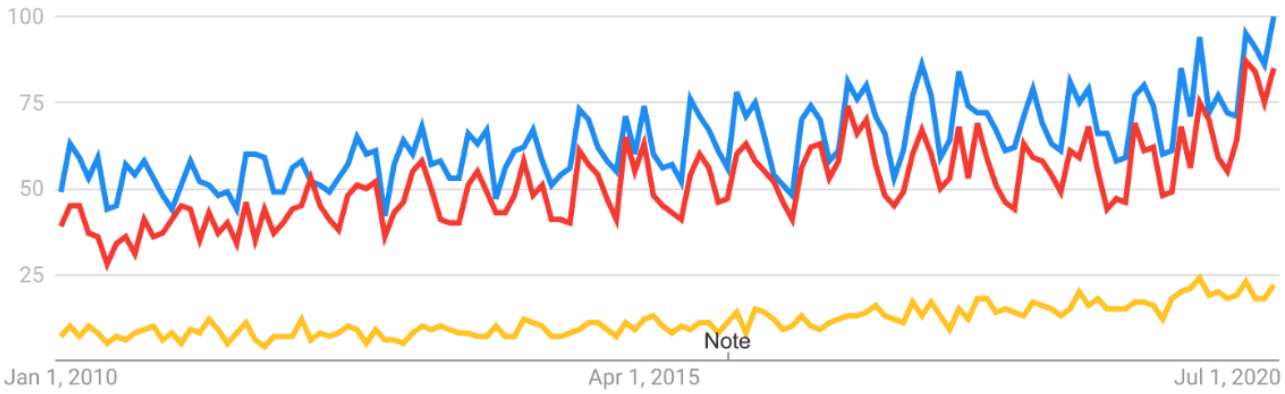
Another change has been an increase in testing after release. This is often dubbed “shift right” testing. Essentially, this just means you test your product in production. There are a number of different ways of doing this. Canary testing involves releasing new features to a subset of users. If there are no problems, the features are gradually rolled out to more and more users. But if there are stability issues or bugs, you can roll back the release. Similarly, dark launching allows features to be launched but to remain invisible to users. This means you can be sure your backend is stable before enabling the new features.

One of the most obvious examples of shift right testing is DevOps. DevOps is a portmanteau of development and operations. But of course, you can’t join those two without testing. Indeed, Bass, Weber and Zhu have defined DevOps[1] as "a set of practices intended to reduce the time between committing a change to a system and the change being placed into normal production, while ensuring high quality". One of the key terms there is “high quality”, as in quality assurance.

1. Bass L., Weber I., Zhu L. DevOps: A Software Architect's Perspective. ISBN 978-0134049847

End to end testing

Shift right is definitely less known than shift left, as the following Google Trends graph shows. However, both are far more common than end-to-end testing. So, why are we talking about E2E testing here?



Relative search popularity of “shift left” (blue), “shift right” (red), and “end to end” testing (yellow) since 2010

Well, end-to-end testing is critical in today's applications, which often depend on external services, multiple endpoints, and 3rd party content. Where continuous testing is about testing throughout the lifecycle, E2E testing involves testing the whole product, including any 3rd party plugins, etc. This is important when you rely on external services such as 3rd party payments or shopping carts, news feeds, etc. Modern applications seldom exist in isolation, so it is vital you test the application within its ecosystem.

The evolution of test automation

Test automation has been remarkably stagnant compared with the rest of the software world. Most test automation frameworks are still recognizable offspring of the original **Selenium**. In particular, they still suffer from the same issues with brittle selectors, test maintenance, and slow test creation. However, things are starting to change.

Selenium is evolving

Ever since its first release, Selenium WebDriver has used a REST protocol. REST works really well for web applications because it allows asynchronous communications. But testing is fundamentally a two-way process. You need to interact with an application and then record the result of that interaction. The latest iteration of Selenium will introduce bidirectional communication. This is similar to how Chrome DevTools and Firefox remote debug work. It will allow you to listen for arbitrary events on web pages, which could transform in-production testing.

Testing in a containerized world

Traditional web apps used a classic LAMP stack. LAMP stands for Linux, Apache, MySQL, PHP. The important point is that backends were monolithic. However, that all changed with the arrival of containerization. People mistakenly believe that Docker invented containers, but of course they existed long before that. However, Docker made containerization mainstream. The thing is, how do you test a system that is actually composed from dozens of independent microservices? The answer is to start automating testing at the functional unit level. That is, don't just test the complete application. Instead, create tests for each unit or container. Then test the bindings between these. Finally, test the application end-to-end.

How AI is powering test automation

One key trend recently has been the rise of AI-powered test automation tools. Artificial intelligence offers some unique benefits that promise to improve test automation. However, these tools are often relatively immature and so need to prove their effectiveness.

Three key approaches stand out:

Self-healing tests

Test maintenance is a huge time sink for most test automation engineers. So, why not use AI to create tests that heal themselves? When a test breaks, a model can analyze what went wrong by comparing the current test run with previous runs. It can then try to find a fix that causes the test to pass again. This is potentially great for tests that break due to UI changes. However, it won't cope so well when the underlying site logic changes.

Model-based testing

Model-based testing takes a novel approach to test automation. Traditionally, testing is done by defining a set of test cases. You then complete these manually, or convert them to test scripts. But what if you started from defining a simple state model showing how your UI works? This is similar to designing finite state machines. The UI is viewed as having a number of states, and a set of actions that allow a user to transition between states. Then an AI algorithm uses this model to define test cases that will test every flow through the UI. On the upside, this will ensure your whole UI is tested. On the downside, it will take significant effort to accurately generate the model. Especially, creating one for the sort of dynamic application we are used to nowadays.

Visual testing

UIs are essentially visual in nature. So surely you can test them using image recognition? Indeed you can. You start by recording your test case. Then your system replicates all the steps involved and compares what is now on screen against the original. If it finds things have changed, it can flag that as a failure. However, there are two things that make this hard to do. Firstly, many UI elements are hidden and only become visible as you interact. E.g. tooltips only appear when you hover over them. But more significantly, many UIs incorporate 3rd party content like advertisements. These change every single time the UI is loaded. Even worse, many other elements of a dynamic UI also change according to context. Not to mention the fun that comes with coping with responsive UIs that constantly resize and reposition elements!

What can we expect in 2021

2021 promises to be a transformative year for testing and test automation. Three things in particular promise to revolutionize test automation: fully autonomous testing, gap analysis, and in-production testing.

Gap Analysis

Modern applications are remarkably complex. UIs are dynamically generated for each user. 3rd party plugins provide embedded content. Many applications no longer have a simple linear navigation. As a result, it's now really challenging to create tests for every single user journey and interaction. The only practical way you can spot where you have test gaps is by monitoring users interacting with your live application. 2021 will see a focus on identifying these gaps in your testing using AI. So, what is needed to achieve this?

Firstly, you need to be able to track how users are actually using your application. This is already done for analytics purposes, but the granularity of the data needs to be a little different. Secondly, you need to understand the scope of your current testing. You need some way of measuring how much of your application is getting tested. This is more challenging than it sounds for precisely the reasons mentioned above (dynamic content, non-linear navigation, etc.). Thirdly, you need some way to compare the real user data and the test coverage and identify the gaps.

In-production testing

As we mentioned, testing has started to evolve. It is no longer simply done just before release. Nowadays, what happens after release is just as important. This is where in-production testing comes into play. Current systems rely on passive monitoring of the backend, combined with techniques like dark launching and canary testing.

The main problem with this approach is that it tends to be too passive. The focus is entirely on performance and responsiveness. Very little attention is paid to the user's actual experience. In effect, you are assuming that the pre-release testing has solved any problems. There's still no reliable way to test whether your 3rd party content loaded correctly. Nor is there a way to test your dynamic UI in production.

In 2021, we will start to see more of a focus on true in-production testing. That means, running your pre-production automated tests against your live system. This will reveal how the system is really performing. However, this is only possible if you are able to test the whole application, including dynamic content. You also need to be able to test cross-browser and cross-platform. Finally, you need to be able to test in a realistic fashion. That means launching tests from locations remote from where your backend is hosted. All too often, production bugs relate to networking issues, which are not spotted if you are connecting directly to the backend. To put it succinctly, 2021 will be the year of QAOps, the logical next step for DevOps.

Full CI-integration for seamless testing

CI/CD is becoming ever more important. It lets companies release new features constantly, so they stay ahead of the competition and react quickly to any issues. However, testing is often a stumbling block for smooth CI/CD. There are two reasons for this. Firstly, a new feature requires a new set of tests to be developed. Secondly, most test automation frameworks require a round of test maintenance to cope with changes to the UI and site code. As a result, there is a less than continuous delivery of fully tested features.

Over the past couple of years, two things have changed in test automation. Tests can now be created almost autonomously thanks to artificial intelligence techniques including natural language processing. And routine test maintenance is almost eliminated thanks to approaches such as Functionize's Self Healing. 2021 will see these features combine to allow truly seamless testing as part of the CI/CD process. New tests can be created as soon as new features are pushed (or even before, using NLP and the test plans). The tests can be initiated automatically by the CI system, and testers can be sure any failures will actually be real. This will significantly streamline the whole process of releasing new software.

Autonomous testing

Autonomous testing is the zenith of intelligent test automation. Two years ago, we published our guide to the **5 levels of test automation**. In that piece, we divided test automation into three phases: test creation, execution and analysis, and maintenance. We identified level 5 as being true autonomous testing. This is where the test framework is capable of creating and running its own tests, integrating with the CI system to trigger tests when needed. Recently, Gartner released their own version of the 5 levels of test automation. According to them, autonomous testing is defined as:

> No human assistance is needed and machines are able to evaluate an application, decide what, when, where and how testing should be performed, and summarize the results to provide a release/no-release decision based on test results.
>
> Gartner Innovation Insight for Autonomous Testing, October 2020.

2021 will surely see the first systems appearing that can achieve autonomous testing, at least for test creation and execution. Imagine if your test tool could monitor your

application, identify gaps, then create and run any missing tests, all without human intervention. This would be revolutionary for many teams. However, it is important to make sure you retain ultimate control and oversight of the testing strategy.

Conclusions

5 years ago, it seemed QA was stuck in a time-warp, relying on test automation tools that were ancient. As we have seen, since then, machine learning and artificial intelligence have made huge advances. 2021 will be a pivotal year for intelligent test automation and will see the first truly autonomous test automation tools. Moreover, it will see testing become fully integrated at every stage in the software lifecycle. The upshot will be better software, released faster, and with fewer wasted cycles.