Using AI to Drive Software Test Automation

Test automation today is difficult, time-consuming, and error-prone. It's difficult because it requires a highly skilled automation engineer to write, edit and maintain scripts. It's time-consuming because making sure those scripts work, and maintaining those scripts as the application changes, typically takes about a man-hour per script.

Further, application complexity is increasing faster than test teams and tools can keep up. Client/server computing has given way to service-oriented architecture and web services, which has given way to web applications and N-tier computing, which is in the early stages of giving way to microservices.

Further, all of these application architectures are actually blends of everything that came earlier. Few applications follow a strict architecture, instead using code and sometimes entire systems from previous applications.

Then there is the larger issue of whether the automation scripts represent what users actually do on the application. While testers sometimes work from user personas and define a prospective workflow, both represent only a guess as to the nature of the user and the path of his or her goals and activities in the application. Real users won't and don't interact in an application only in the way test scripts are written.

To be effective, automated testing must accurately represent how real users interact with the application.

Today, test scripts are typically recorded or written by testers, who go through multiple steps within the application. Often these steps are meant to exercise individual pages and functions rather than replicate a user workflow. That can be a decent start, but errors often show up only when tasks are performed in a specific sequence.

Newer development methodologies are also uncovering the limits of test automation. Both agile methodologies and the newer DevOps trends demand fast and high-quality automation to keep up with an ever-increasing pace of rapid development and continuous deployment. The technical difficulty of automation today, coupled with the need to constantly modify test scripts to keep up with the software, prevents testing from being as responsive, fast and thorough as it needs to be.

The end result is that automation, despite its potential in improving quality and accelerating test, often fails to live up to that potential because of limitations with existing automation tools and techniques, most of which have their grounding in 30-year-old methodologies. Many organizations that attempt test automation often give up or relegate it to a small role because it doesn't turn out to be a silver bullet. Those that are successful with test automation end up devoting a large amount of human resources to it.

To be effective, automated testing must accurately represent how real users interact with the application, and the ability to rapidly create test cases based on those steps. Further, these test cases have to be easily maintainable and changeable as the application changes. Representing actual use improves software quality by focusing on the workflows that matter.

Enter AI and Machine Learning

Fortunately, this type of problem is amenable to a solution using artificial intelligence (AI) and machine learning. Testing teams are turning to AI-based solutions such as Appvance IQ, not to replace human testing processes, but to dramatically augment them. In particular, AI scripting largely replaces the mundane tasks of manual scripting, with a knock-on benefit of reducing the need for manual testing. This allows team members to focus on important areas

that today often get ignored: e.g., application usability, flow, layout, test analysis etc. In short, allowing an AI system to "write" test code automatically can be a godsend to an organization.

Inputs to AI Scripting

To automatically generate tests that are meaningful, one must consider the inputs that would be required. At a minimum, and assuming server level tests (such as HTTP or API tests), six functional areas of inquiry must be addressed.

1. How do users use or will use an application?
2. What are the expected results and/or responses? How do we know a pass versus fail?
3. How does the application function and what is its purpose?
4. How are requests formed so they will not be rejected by the server, even if the server is now a different server (for instance the QA server instead of the production server)?
5. What data is required for valid forms (such as credentials)?
6. How can correlations be created to take server responses and place them in future requests (such as session ID's)?
7. How can changes in a new build versus the old build be handled?

The problem is not a simple one since the AI scripting system must have sources for, or the technology to create, all of this data, then learn from the data, then utilize that knowledge to create new tests. It is complex and a machine-learning big-data challenge of the highest order. And of course, you must also repurpose those use cases into different types of tests (functional, performance, security) based on their functions and goals.

Operational data can be quite useful as an AI learning source.

Fortunately, virtually all organizations collect data on their applications, as well as the application environment in which they run, such as the server and network. While often ignored, this operational data can be quite useful as an AI learning source. For some shops, it may be as simple as server and web logs. Others may have third-party tools, such as Sumo Logic, Dynatrace or Splunk, that collect supplementary data or do analyses on existing application and network data.

This data, which records limited information (e.g., unique user IDs, their flow through the application, and results returned, latency, "thought" time during navigation) can be used by machine learning to build a model of user activity. Even better, it can be used for more than that, as we shall see.

Response codes may also be captured, so the AI scripting system can rapidly flag bugs found by the system but unreported by users. These can be used to create tests to repeat those bugs, and then run them to validate the current state. This "rapid bug ID and Test" capability can allow teams to repeat the conditions that caused the failure in the first place, and then turn around fixes in minutes, rather than hours or days.

For question #2, you can also look to server logs to understand expected results based on prior results, or the user can add specific validation for any page or response broadly across thousands of use cases. Validations can also be data-driven, in that they can employ test data to deliver results that accurately reflect user activities.

For #3 and #4, it's possible to employ several methods to better understand the way the application works and how to form requests and expected responses. For instance, Appvance IQ's AI Scripting uses algorithms to create an Application Blueprint that attempts to locate

every possible (and practical) page or response in an application. This teaches the system about the application as well as proper request formatting.

For #5, valid test input data (e.g., credentials or user data) is provided by the test team or by a connected database. The system should easily make use of the data in the right positions.

For #6, the system must create errors on hidden runs to search for appropriate matching substitutes that would pass or correlate based on a score of likelihood of matching the requirements for a future request. Automatic correlation is required in order to send accepted requests to the server across thousands or millions of requests, often in parallel.

To address #7, Appvance engineers developed a system to automatically note flow changes in a new build versus prior builds, and then apply that learning to script generation. Thus, for many or most cases, script maintenance becomes labor-free and completely machine driven.

Applicability of AI Driven Test Automation

AI-driven test generation is applicable to many application types: websites, web apps, backend services, mobile apps and IOT devices, as well as various test types such as functional, performance and security. As long as there is user and application data, it can be applied almost anywhere.

AI-driven test automation creates and executes thousands of user-driven scripts on-demand.

The result is an AI-driven test automation system that creates and executes hundreds or thousands of user-driven scripts on-demand, based upon learning from various rich data sources, and attain extremely high test-coverage with minimal human intervention. Further, these portfolios of AI created scripts can drive functional tests and data-driven load tests. In short, AI scripting reduces the labor cost of script generation to zero.

Essentially, the Appvance IQ's AI scripting is an automatic code generator. It's smart code in a very bound case (testing), but code nevertheless. No need for Selenium or Java or QTP or whatever your scripting language of choice would have been. Because, manually scripting will become used only in limited corner cases, allowing the AI system to handle the vast majority of code coverage and generating tests which far better represent actual user interactions.

AI Driven Test Automation Is Available Now

Test automation doesn't have to be as difficult as it is today. By utilizing the information already available, test automation teams can bring AI technology in house to accelerate test script development and simplify the script life cycle.

Appvance IQ is available now. Let's talk soon about identifying an attractive application it can test as a proof of concept.

Appvance, Inc.
3080 Olcott Street
Santa Clara, CA 95054
Appvance.ai