SUBJECT7
QUALITY IS A CULTURE

Wait, Is Avoiding Low-Code an Automation Anti-Pattern?

By Paul Grizzaffi for Subject7

I've said it. You may have heard me say it, or at least seen me write it. Automation development is software development, and we need to treat it as such. I've also said the use of record-and-playback, drag-and-drop, or low-/no-code tools is generally not a good way to create an "enterprise-class" automation endeavor.

The reasons are myriad, but they usually filter down to this: in many cases, the cost of maintenance of an automation endeavor outweighs the value you get from "fast" script creation. When an interface changes, regardless of whether the interface is an API or a GUI, maintenance is required to keep the automation in parity with, and providing value for, the application or product. It's important to note that maintenance for application parity is not unique for a "non-code" tool; parity maintenance is required for all types of automation. Despite what my Ronnie James Dio autograph says, magic doesn't exist in software development.

When discussing no-code tools and frameworks, the executable artifacts that are traditionally produced from these kinds of tools are difficult to modify and thus often require re-recording (or the equivalent thereof) to maintain parity with the application they are responsible for aiding with testing. To be fair, this sector of testing tool development has progressed in the last few years to make its artifacts more maintainable, but it usually pales in

comparison to automation directly created via a traditional programming language such as Java, C#, or Python (yes, there are others, but I didn't want 50% of the article to be a list of "programming languages for test automation").

Now, as metal as I am about automation and software development, I am a bit of an outlier when it comes to low-/no-code and record-and-playback. As I discuss in my talks *Myths About Myths About Automation* and *Not Your Parents' Automation*, low-/no-code and record-and-playback technologies have their place. In my experience, they tend to provide the greatest value in two contexts:

- When the interface and behavior against which we are automating rarely changes.
- When the automation being developed is disposable, i.e., it will be used for a specific, short-term purpose.

Note that in both contexts, maintenance is not a major concern, so the downside of "effort-intensive maintenance" doesn't play a factor in choosing an automation tool.

There is, however, a huge assumption built into most of what I wrote above. That assumption is that the teams creating and supporting the automation either have programming and automation experience or they plan to acquire that experience. This is not always the case.

Most people reading this are probably saying, "What do you mean? How are you going to deliver software without some abilities in these areas?". Let's consider that most users don't want software; they want a solution to a problem. They want information faster or with higher quality, they want products to be delivered faster, and they want less friction in achieving some goal. They do not want software; in most cases, they have accepted that software is (currently) the best way to accomplish their goals.

As a supplier of solutions to a set of users, clients, or customers, some organizations may need to deliver software; again, software is often the necessary evil. That said, not every organization is ready to be a software delivery organization. Some organizations deliver software solutions out of necessity, even if that delivery is outside of their core competency. This overall direction is often set at a corporate level: if "corporate" decides software delivery isn't a core competency, then it's not. Those teams need to trust that this is the appropriate business decision, at least currently.

Consider the case of a provider of financial services. This provider creates minimal software of their own. They create conduits to amalgamate several 3rd party software applications to fill their clients' needs. They probably have minimal programming staff, and that staff may be creating those conduits as part of their job, not as their whole job. This kind of team may well be counting on the conduit developers to perform the testing of those conduits as well as develop them. Must this kind of team learn even more programming concepts to create and maintain test automation to benefit from software-based testing assistance? In the larger scope, no, they should not have to do so. This is outside their core competency. Perhaps, low-/no-code test automation might be of value in this kind of situation.

Perhaps we need to add a third bullet to the bullets above that states: When the value provided by low-/no-code automation exceeds the cost of creating and maintaining it.

That "third bullet" is a specific example of a concept that I mention in several of my talks, namely that "something is better than nothing, as long as the something is providing sufficient value". I usually couch that statement in the context of "if you've gotten X% of a task or flow automated and the remaining percentage may be more effort than it's worth, perhaps you should stop at X%". It's a simple extension of that notion to say, "if low-/no-code automation is providing value, why not embrace it".

Make no mistake. Even low-/no-code and record-and-playback create software. We may not see it, but somewhere, under the hood or behind the curtain, there are software outputs and artifacts. The tool produces some item that represents what has been created for a given script, flow, or activity. That needs to be stored somewhere. After that item is executed, there is (hopefully) a detailed account of what happened during that execution such as logs and results; those artifacts also need to be stored somewhere. Those scripts, flows, or activities probably need to be executed in an unattended manner such as in a CI/CD/CT pipeline; the items created by the tool need to be available to be executed in such an unattended manner. Low-/no-code and record-and-playback tools are not magic; they are rooted in software development and their applicable pieces need to be handled as such.

To be clear, I'm not being hypocritical. I stand by my general assertion that most software delivery teams are appropriately suited, or can be, to create "programming-based" automation to assist in their testing; recall that in most cases, the cost of that maintenance and upkeep is lower than re-recording test flows or attempting to maintain artifacts generated by a tool. That said, much like the aforementioned financial services provider, automation creation is not in every team's core competency. Every hour a team spends doing something that doesn't directly and positively impact product delivery might be an opportunity cost.

Additionally, decisions we make today about our test automation don't have to be written in stone. Do we upgrade our app? Do we use new frameworks, concepts, and implementations? Of course, we do! Similarly, we should not be reticent to do the same with our automation. We make decisions at a specific point in time and with limited information; we do our best. In the same way, organizations watch the evolution of tools in their application development frameworks, we must do the same in our testing and automation frameworks. Certainly, changing automation frameworks is a cost, but that change might also provide great benefits. We won't know unless we observe and research.

Please take note of my previously used weasel words: general, most, and typically. Context is king, queen, and guillotine. Teams must adopt and adapt what is valuable and appropriate to them, keeping an eye on their core competencies. If we don't adapt to the changing contexts, we end up in the quagmire of missed expectations and loss of value.

Like this? Catch me at an upcoming event!

Learn more about how Subject7 is Unifying Test Automation with Codeless Automation that accelerates test authoring, reduces test maintenance, and scales in the cloud or on-prem with enterprise-grade security.

SUBJECT7
QUALITY IS A CULTURE