Dev and Test - Switching Lanes?

By Matt Heusser for Subject7

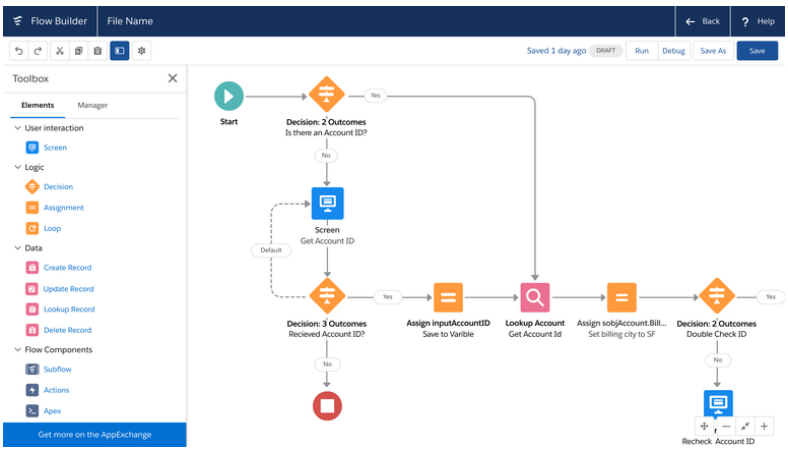
In computer science we strive for more and more power. The way to do that is typically through reuse and high-level logic. Write a function, call it, reuse it, never write it again. In other words, Don't Repeat Yourself (DRY). Sometimes it can be helpful to understand what is going on at lower levels. For me that was one college course in Assembler and a graduate course in computer architecture. Besides those two courses, the key to power was always to work at higher and higher levels. That trend continues in low code tools for programmers. Yet as programmers, we are getting away from code by moving to drag and drop; testers seem to be going the other way, learning to code, and expressing tests in "regular" programming languages like C# or Java.

What is happening here?

Developers and Low-Code Tools

It's easy enough to create a simple Create Read Update Delete (CRUD) application based on database tables. In Visual Basic you can get a database control and connect text fields; in Ruby on Rails, you give the table a name and have the application render itself. The low-code tools take things to a new level. There is often no software to install; the low-code tool runs in your web browser in the cloud. Upload a spreadsheet and the software will create a database and website for you. Add your company CSS to automatically style it. For simple, straight-forward workflow automation, low-code tools are gaining a place. You can even drag and drop complex flows. That might include selecting flight, seat assignment, adding credit card information, then clicking to change only the return flight information.

The example below, from Salesforce *Lightning*, shows the drag and drop interface how screens are connected. These ideas have been around in development for decades but are finally achieving acceptance. Mendix, the leading low-code platform, had $100 million in annual sales in 2019, representing only a fraction of the market. Microsoft, Oracle, and Salesforce all offer tools that integrate deeply with their traditional software tools yet provide drag-and-drop, from the cloud functionality.



And then there is testing...

Once Bitten, Twice Shy

In software testing, we tried the high-level tools approach. We called it record & playback. Those days currently exist in the back of some testers' minds, in a time period one might call the paleolithic. That is so far back that very few practitioners even remember. That was the time when the articles like Hey Vendors, Give Us Real Scripting Languages or Test Automation Snake Oil were written. I am talking about a different *century*, when software was *burned on disks* and sent in the mail to run on Windows. Windows itself was a graphical hack on top of MS-DOS not designed to be "driven" by any sort of tool. There was a complex build pipeline that kicked out a Windows application. Testers could set up the data and run the application on their desk during lunch, or else not work. Personal computers were expensive; laptops were for executives.

Most testers of today weren't practicing their craft at that time. What has happened is that the thinking of testing communities, to the extent that we have them, was heavily influenced by people who read those articles, or even influenced by the people who were influenced by people who read the articles - and so on. Along the way we have lost more than a little context, which is why I like to point back to those tools. Around that time, I was programming in Visual Basic myself, which was incredibly powerful at doing a very limited set of things, and incredibly frustrating once you exceeded what it was good at. Too many testers look at low-code, and "know" that "we tried that trick, and it didn't work."

The alternative is to go deep code - to write in a production programming language. That gives us the problem that James Whittaker pointed out at Google's Test Conference in 2010; trying to write a program on top of a program that is changing. That means testers spend a great deal of time programming, debugging, and maintaining test code, instead of focusing on exploring and reducing the risk in today's changes. To put it differently, we saw a problem two decades ago in record-playback and, in some circles, the pendulum swung too far the other way. The question to ask is: with a rich ecosystem and flexible tooling available, is heavyweight coding still the correct option for your organization?

Programmers are always reaching for more powerful, higher levels of building software. This is a meme-level conversation in the programming world. People write jokes about it with images, like this one.

It is possible that some testers have "programming envy." They see programmers as getting respect and honor by automating things with code and want to repeat the behavior. Certainly, at some of the programmer-first companies, like Google, the way to promotions, credit, and respect is through code. Given all this, it is understandable that testers might be attracted to "high code" solutions.

Four Choices

Taking a step back, I see that programmers can use traditional or low-code tools; testers can create work with traditional or low/no-code tools. That creates four options. Let's talk about them.

		TESTER PREFERENCE	
		Low-Code	**High-Code**
DEVELOPER PREFERENCE	**Low Code**	1	2
	High Code	3	4

1. **D*eveloper Low-Code/Tester Low Code***. Some of the low-code frameworks are providing their own low-code test tools. These tend to demo the software, to create a sort of living documentation, and to show how the software should work through execution. If these tools are tied together in the same platform, there can be a little bit extra potential, as the test can reuse libraries from the tool. If the tool does not have an affordance for testing, the application may look like any other web application, and adaptive tools may be appropriate.

2. ***Developer Low-Code/Tester High Code***. This seems unlikely. In the simplest of applications, there is just no need for the power and flexibility that writing code creates. In this environment writing and running the tests may take longer than writing the production application!

3. **Developer High-Code/Tester Low Code**. The same combination the test community was critical about ten years ago has grown up. There is something to be said for being able to view a test in business terms, yet drill in for detail. This sort of test work can be done by analysts who are less technical, read and understood by product management, and act as a sort of working specification of what the software should do.

4. ***Developer High-Code/Tester High Code***. In this environment it is possible for the testers and programmers to program in the same language. The two groups can use shared libraries, can blur the line between test and code, can check the test code into the same place as production code and use the same branches. This use of the same branches minimizes the problem of trying to run "new" tests on "old" code, for example, verifying a small change that is a production patch.

Conclusions

As we have added web, phone, tablet, watch and other form-factors, we keep pushing the software test tool industry backwards, from maturity to growth. That is fantastic as the number of valid test approaches keeps expanding, now to include writing code, record/playback, and now adaptive testing tools.

When I started my career, programmers were writing sort functions in C++; today they use a vector object with a .sort() built in. The habit of programmers is to look for more powerful programming languages with less custom code. Today's low-code trend took decades to arrive and had several fits and starts. The same sort of lesser-code tools exist for testing and promise to reduce the code-testing-code, or "two targets" problem. While these tools appear "new," Mercury's WinRunner software debuted in 1995 - making them about twenty-five years old.

There's no need to wait. The waiting has been done. What are you waiting for?

Ready to learn more about Subject7?
Contact us today to request a free demo.