Less Is More: Picking Your Test Automation Language

It's a classic dispute: Two test automation engineers can't agree on which programming language to use. In some contexts the strong points of a certain language definitively make it the right choice, but what do you do when either language could work well for a project? That's when it becomes a managerial decision.

It was years ago, while working as a chip-level test engineer, that I first I witnessed an argument that has repeated itself a dozen times since. Two test automation engineers could not agree which programming language was better for their next-generation automation system: Pascal or C? (I did mention that it was years ago, right?)

Each side had weighty arguments: what's easier to do in each language, how important strong string-processing is, what language has better libraries, etc. Additionally, both sides claimed that there isn't really a great need to align on one language—it is possible to write modules in one language and use them as libraries for the other language.

The arguments continued for a while. If my memory serves, the automation was mostly written in C for a simple reason: It was the preferred language of the developer who wrote most of the code.

There are cases where the decision is clear-cut and easy, because in some contexts the strong points of a certain programming language definitively make it the right choice. But what do you do when this is not the case, and each language has a list of pros and cons?

Years later I was consulting for a team of testers writing a software test automation framework. Except for the fact that the languages discussed were now Visual Basic and C#, the rest was very familiar: the endless arguments; the teleconferences and face-to-face meetings, vowing to continue until we reach a decision; the presentation of proofs by each side about why their position is the correct one—all while employing both languages as each side tries to generate enough code to weigh the decision so that their language is chosen.

Who is right?

It seems to me that everyone is right … and no one is right.

The claims on each side are usually correct, which means you can't settle the argument based purely on technology attributes. Comparing the pros of one language to those of the other is really an argument about the reasons for both languages to exist. Clearly each language has some aspects where it is better than the other. When you compare

the positive and negative aspects of any language, it is hard to judge which parameter is more important. With no single strong argument that tilts the scales in favor of one or the other, the discussions drag on an on.

In these situations, I think it's important to realize that the main consideration should be the number of languages, not which language.

Picking a Language Is a Managerial Decision

There is an "activation energy" for each new language used by a team. When context-switching from one language to the other, the engineers lose some time recalling the technical details of the language. You may need to duplicate some of your development tool (e.g., IDE, code control) and support systems, such as your nightly build and continuous integration. The engineers will need to learn how to work and be proficient with more tools. All this takes time, which costs money.

Engineers already are not very keen on reusing someone else's code. There is always the feeling of "It will take me less time to just rewrite the whole thing from scratch." The chance of code reuse when that code is not written in the language the engineer prefers approaches that of pigs flying.

When you need to transfer ownership of code between engineers due to a career move, role change, or the proverbial bus, you will find that the language used to write the code narrows the list of possible candidates. You will have a harder time recruiting as well, as you will be looking for candidates that know both languages fairly well. Alternatively, you will need to pay for the new hire's learning curve.

The conclusion is that in many cases, the decision on a programming language is not an engineering decision, but rather a managerial decision.

The manager should listen to both sides to see if indeed this is a situation with no clear argument for one side. If that is the case, the parameters to consider are managerial:

- What support tools does the organization use?
- What language is known by more engineers?
- What language is the tested code written in?
- What's the preferred language of the person who will write most of the code?
- What's the language used for more of the existing code?
- What language is used by systems that need to interface with your code, and will you need to update those systems?

Of course, there also will be other, situation-dependent (nontechnical!) factors.

Note how these questions do not try to decide if one language is technologically superior to the other; they sidestep that issue entirely. These are objective questions that in most cases are easy to answer. If any of the answers does not help in the decision—such as, there is no one language that significantly more engineers are familiar with—you can just drop this aspect from the list.

But what if, even after evaluating the above questions, there isn't a clear choice? At this point, it doesn't matter which language you choose; the important thing is that you do choose one, because the delay may be more expensive than the cost incurred if you made the wrong decision. As the band Rush once sang so well, "If you choose not to decide, you still have made a choice."

So flip a coin, get the team to back whatever the outcome is, and follow through with supporting that language.

First published in Hebrew on "Testing World" magazine (2nd issue, July 2015)
http://itcb.org.il/index.php?option=com_k2&view=item&id=754-מגזין-עולם-הבדיקות-3-2015&Itemid=865גיליון-שני-רבעון

Published: December 28, 2015
http://www.stickyminds.com/article/less-more-picking-your-test-automation-language