Migration

Adrian Kosmaczewski

2007-01-28

I came accross this interesting posting on The .NET Addict's Blog. It is interesting to read since it comes at a time where "traditional" Microsoft developers get interested in other technologies, such as Ruby on Rails, Cocoa or Linux. And the same can be said about Java, where lots of luminaries like James Duncan Davidson (the creator of Ant and Tomcat) started moving towards other technologies (in his case, this happened at the beginning of this decade even).

Traditionally Microsoft has had a very strong relationship with their developers, since they are the ones that create the applications, that ultimately drive the sales of Windows, whatever the version. But more and more, developers are discovering new ways to do things. And when developers open their minds, it means that the next generation of project managers, architects and CIOs will change the shape of industry; this is a slow process, but a steady one. At the end, some technologies will prevail, others will fail.

There is a big change going on. But what are the common traits of the technologies that attract developers?

I will concentrate my comments on programming languages, which are, after all, the basic tool we use every day in our job. Of course, IDEs, code generators and frameworks usually come after, bringing productivity and ease of use, but they usually depend on the underlying programming language.

The first common characteristic of the languages that are attracting developers is dynamicity. If you compare Ruby, JavaScript, Objective-C, Smalltalk and Lisp, with C++, Java, and the .NET languages, you see that the new languages allow you to dynamically interact with objects on the runtime environment; it's not reflection-by-API (like in the case of .NET or Java) or reflection-by-macros (like you need to do in C++) but direct reflection, embedded in the language and the runtime. Just ask the object about its class, add some methods to it (even if you do not have the source code of the object!), create the API that best suit your code, and make your code as readable as a Paulo Coelho book.

Another benefit of dynamic languages is the fast write-compile-run cycle; in most of today's fashionable languages, the lack of a compiler helps delivering faster, not only at release time, but also at maintenance time (which far longer).

Even in the case of Objective-C, which is a compiled language after all (thanks to the GCC suite), most of the references are resolved at runtime, so you have faster compile-link cycles actually; Objective-C is an extremely interesting language, that has a feature that I haven't seen anywhere else: the id type. id is not NSObject (the root class of Cocoa), but rather a "placeholder" type (and more similar to a void* type, actually, but easier to use), which allows a variable to point to whichever object, no matter the type; this allows you to be extremely dynamic, like in scripting languages, while at the same time being able to optimize your code using static references.

However, nowadays, the only factor where static typing bring great value, is in the case of IDEs: autocompletion and "intellisense" are all empowered by strong typing, and thus, when you hit the "dot" character, you get the list of methods of the object you are working on. Usually performance was also considered a byproduct of static typing; but what about nowadays? Yes, compiled languages perform faster, but economical and market reasons are making dynamic languages more interesting everyday; code is much more than performance. Time-to-market, maintainability and readability are getting more important today. And, besides, faster processor speeds make runtime languages an interesting and fast option, after all.

The second common denominator is automated memory management. It just frees your mind and time to think about the problem in hand, instead of having to worry about dangling pointers and memory leaks. Actually, this was the missing element in Objective-C, but Apple will be incorporating it in version 2.0. This makes Objective-C an even more interesting language to developers: you get performance (you can use static typing if you want, which also empowers the Xcode IDE), you get flexibility (you can use pure dynamic typing if you want), and you get automated memory management (which lowers the overall workload of a developer). Will Objective-C's garbage collector bring more enterprise developers to the Mac OS X platform? I think yes.

The third common factor is the reduced lines of code that you write in this language, compared to the statically-typed ones (this is kind of a corollary of the second factor I've mentioned above). And simply stated, less code means less maintenance costs, faster bug resolution, and faster time-to-market. These are important factors today; and one of the marketing factors that make Ruby on Rails so popular. This will not mean the end of statically-typed and system languages; particularly C++ and Java have a strong future in the finance industry, where they still are very strong.

In conclusion, I think that the trend is very well set, and that the migration is taking place. Developers are migrating towards dynamic, garbage-collected, and more discrete languages. The benefits are there, and now, the performance too. Remember when EasyJet started? Nobody imagined that a small airline selling on the web could overtake the industry. It is, as Paul Graham says, the Power of the Marginal. And now we'll see that in the programming world as well.