Musings of an "Old-School" Programmer

Stephen B. Jenkins
Institute for Aerospace Research
National Research Council of Canada

I admit it; I'm an old-school programmer.

If you've never heard the term "old-school" before, here's my take on it. A near as I can tell, the phrase had its origins in early twentieth century expressions such as “a gentleman of the old school” and “old-school ties”, where it was meant to bring to mind the traditional ideas and social structure of the upper-middle-class in Britain. Recently, it has come back into vogue, beginning with the music world – particularly hip-hop – where it has been used to describe the 'traditional' form of rap music dating from the late '70s and early '80s. Since I personally have no connection to the rap music scene, I first heard the term in reference to the building of custom motorcycles. In the biker world (a community with which I am well acquainted), "old-school" refers to the style of choppers made during the '60s and '70s. Typically, the builder would start with a stock Harley-Davidson, Triumph, or Norton, and remove all the parts that weren't essential to getting you moving or getting you stopped. This meant "chopping" off the windshields, saddlebags, and crash bars, and "chopping" up and modifying the frame, fenders, gas tank(s), oil tank, seat, forks, and handlebars. (In more philosophical terms, they adopted a minimalist, "form follows function" approach.) All of this work was done by hand, using traditional tools, techniques and materials. For modern builders making old-school style bikes, this means no CNC milling machines, no water jet cutters, and no fiberglass or plastic. Recently, I've begun hearing the term "old-school" quite often during television programs such as "American Chopper", "Biker Build-Off", "Motorcycle Mania", and, with reference to hot rods, "My Classic Car".

So, what, you may ask, does all this have to do with programming? Well, I believe there are many similarities between building custom bikes, and building custom software. (This seems to be especially true for developers working in small software shops or as solo contractors.) Both endeavours are a complex combination of art, craft, and engineering, with work that is frequently difficult and sometimes frustrating, but usually fun, and occasionally exciting. Both fields require significant creativity, and an ability to visualize, from the beginning, what the final product will look like. There are several concerns that are common to both types of builders: reliability, performance, maintainability, and style (AKA "look and feel"). Both activities seem to engender a very strong sense of ownership and professional pride: "You can say what you want about me, but don’t say anything bad about my work". Finally, those who excel in either area often appear to posses an almost intuitive knack for knowing what solution will work best in a given circumstance; they make decisions based on a “feel” for the problem at hand.

As I said in the beginning, I consider myself to be an old-school programmer. By that, I mean that I have an approach to design and development that is similar to that of the old-school bike builders. I think this manifests itself primarily in the tools I choose to use (and, by extension, the tools I choose to *not* use), and the way I work. That's not to say it has no effect on the final product, just that I think it's most noticeable while I'm designing, developing and debugging.

The Tools I Use

My number one tool is a powerful editor. Although my personal favourite is Vim, Emacs seems to be the other, best option. Either of these two will give you as much power as you need, and they both work well with all languages and modern operating systems.

My number two tool is a good shell (I prefer bash), with a powerful set of utilities that I can use from the command line (e.g., awk, diff, grep, and Perl). Before anyone accuses me of being Unix-centric, I'll just point out that I use these same tools on Windows XP, and I know that they're available for OS X as well.

The third, and last, tool I want to talk about is knowledge; specifically, an in-depth knowledge of the problem domain and of several types of programming languages. In the same way that you wouldn't want a bike or hot-rod built by someone who only knows how to work with mild steel (they should have experience working with stainless steel, aluminium, brass shims, and copper wires), you should be cautious of a programmer that only knows one language. You should be more than merely cautious of someone who only knows how to program in one language, on one OS, using one IDE.

You'll notice that this is the first mention of IDEs, and that I've said nothing about UML or other CASE tools. That's because I don't consider them necessary; in fact, for myself, I find that they hinder more than they help.

Originally published in *Communications of the ACM*, Volume 49, Issue 5, May 2006.

The ROI is way too small to justify the overhead incurred; you end up spending time fighting with the tools instead of working on your code. And speaking of working….

The Way I Work

During the preliminary, top level design, I tend to spend most of time just sitting and thinking. In fact, I seem to do my best design work at the local coffee shop, rather than at the office. I usually just use pen and paper, although I have been known to resort to a whiteboard if there are others involved in the design process. I typically make a quick sketch of the user interface, and for large projects, a diagram of the interaction between the major code blocks. After this brief initial phase, I don't really see design and development and debugging as separate tasks; instead, each happens as needed while I'm writing code. I usually build a basic framework for the program, and then hang everything else off of it, making sure everything fits properly, works well, and "feels" right. You'll notice this is not unlike building a bike by starting with the frame, then adding suspension, wheels, engine, gearbox, etc., one at a time, verifying things as you go.

If you thought my views on IDEs were heretical, wait until I tell you what I use for debugging: print statements. Yes, seriously: print statements. Once or twice a year, I may need to fire up a command line debugger, but generally, print statements and log files work just fine. They are language and OS neutral, and work extremely well with my preferred programming style: a very rapid write-run-debug cycle (typically fewer than 10 lines of code per iteration). In some complex environments, such as developing Perl CGI programs to dynamically create JavaScript code that generates CSS and HTML, using print statements and error logs is the only way to debug.

Concluding Remarks

Now, before the IDE and UML folk begin to light their torches, grab their pitchforks, and storm up to the castle to do away with the purveyor of such monstrous ideas, I want to make it clear that I'm not saying that this is the only right way to do things. In fact, I'm not even saying that it's the best way, just that it's the best way *for me*. And maybe for other people in similar circumstances (i.e.: small, extremely dynamic, custom software development groups). Although I don't know this from personal experience, I believe that old-school programming is rarely practiced within companies that have large teams of programmers working together on projects. This is not, I suspect, due to the techniques being less valid in those settings, but rather it's because their managers seem to prefer methodologies that give them tighter control over their staff. This, in turn, causes developers who favour old-school ways to go elsewhere. (I know that if I found myself placed in such a situation, I'd quit and seek employment with a small R&D group. That's just another part of the old-school mentality, I think; I'd rather change jobs than be dictated to about how I should work.)

Please note, this article is not an "old is good – new is bad – those were the days" rant. While I may be an old-timer (I wrote my first program on an IBM mainframe in '74), I'd never want to go back to the days before dynamic languages. And I don't spend my time churning out spaghetti coded FORTRAN or COBOL on a VT100 terminal – mostly, I develop web-based applications for aerospace research [Jenkins]. I've used an old-school approach over the years with many languages (I've just started playing around a bit with Ruby) and several development styles (I currently use an agile methodology) and it's never failed me yet.

I know a few old-school programmers, and, like old-school bikers, they don't seem to care what others may think about the way they work. They just want to do their own thing, their own way, and in doing that, produce something of which they'll be proud to say "I made that!" If you are an old-school programmer, don't apologize for it. Just keep putting your efforts into the things that help you produce the best code possible, and ignore all the bells, whistles, and eye candy that makes those graphical IDEs so flashy. Remember, as my biker buddies say: "chrome won't get you home".

Reference

Jenkins, S.B., *A Web-Based Environment to Support Aerodynamic Testing*, IEEE Aerospace and Electronic Systems Magazine, Vol. 19, Num.1, Jan. 2004, pp. 3.

Bio

Stephen B. Jenkins is the senior programmer/analyst at the Aerodynamics Laboratory of the Institute for Aerospace Research, National Research Council of Canada. He rides an '87 Harley and a '56 Norton.

For more information, go to www.erudil.com