BLOG@CACM

The *Communications* Web site, http://cacm.acm.org, features more than a dozen bloggers in the BLOG@CACM community. In each issue of *Communications*, we'll publish selected posts or excerpts.

twitter
Follow us on Twitter at http://twitter.com/blogCACM

DOI:10.1145/2507771.2507775 http://cacm.acm.org/blogs/blog-cacm

Helping Scientists, Engineers to Work Up to 100 Times Faster

Philip Guo explains how programming skills can make scientists and engineers more efficient and creative.

Philip Guo

Why Scientists and Engineers Must Learn Programming

http://cacm.acm.org/blogs/blog-cacm/166115-why-scientists-and-engineers-must-learn-programming/fulltext

July 18, 2013

In recent years, there has been an admirable push to get more people to learn programming. But if I have never been exposed to programming, why should I invest all of the effort to learn? What is in it for me?

Pundits often give fuzzy responses, like claiming that programming is the "literacy of the 21st century," that it helps you become a more empowered citizen, and that it enables you to create magical works of pure creativity.

Even though I agree with many of those claims, I am not convinced that they are concrete enough to motivate someone to devote the thousands of hours necessary to get proficient at programming.

Instead of trying to convince everyone to learn programming, I have a more modest goal: encouraging scientists and engineers. Here is my value proposition to them:

If you are a scientist or engineer, programming can enable you to work 10 to 100 times faster, and to come up with more creative solutions than colleagues who do not know how to program.

Kevin's Story

Modern-day scientists and engineers are spending more and more of their workdays in front of the computer. As an example, consider my friend Kevin, who works in oceanography and mechanical engineering. Whoa, sounds like he is probably spending all day out on high-tech boats, rigging together mechanical devices like MacGyver and collecting data from underwater sensors, right? This must be his typical workday — hard hats and heavy-duty work gloves.

Actually, Kevin spends less than 5% of his time out on the ocean; the other 95% of the time, he is sitting in front of the computer writing programs to clean up, transform, process, and extract insights from data collected out in the field.

The same story plays out for scientists and engineers in all sorts of fields: astronomers, biologists, physicists, aerospace engineers, economists, geneticists, ecologists, environmental engineers, neuroscientists...the list goes on and on. Modern-day science and engineering is all about processing, analyzing, and extracting insights from data.

Three Reasons to Learn

Over the past few years, many scientists and engineers have ranted to me about how furious they are that nobody made them learn programming back in high school or college. They now realize how much more productive they could be at work if they had developed those skills earlier.

Based on these conversations, I've come up with three reasons why scientists and engineers must learn programming:

1. You can work 10 times faster by writing computer programs to automate tedious tasks (such as data cleaning and integration) that you would otherwise need to do by hand. If you know how to program, computer-related tasks that used to take you a week to finish will now take only a few hours. I cannot think of any other skill that leads to an instant 10-times productivity boost for scientists and engineers.

2. Programming allows you to discover more creative solutions than your colleagues who do not know how to program. It lets you go beyond simply using the tools and datasets that everyone else around you uses, to transcend the limitations that your peers are stuck with. For example, you will be able to write programs to automati-

cally acquire data from new sources; to clean, reformat, and integrate that data with your existing data, and to implement far more sophisticated analyses than your colleagues who can only use pre-existing tools. By doing so, you are more likely to make a creative innovation that your colleagues would not even think of exploring due to lack of programming skill.

3. Finally, knowing how to program allows you to communicate effectively with programmers that your lab hires to do the heavy-duty coding. I do not expect you to become as adept as the professionals, but the more you know about programming, the more you will be able to relate to them and to command their respect. If you can motivate programmers in your lab to spend more of their time helping you solve technical problems (e.g., by writing parallel programs that run on a compute cluster), you can work 100 times faster than if you had to attack those problems alone.

Thoughts?

Email philip@pgbovine.net

Postscript

Readers have responded with two main classes of comments:

1. Scientists and engineers in lots of fields already learn some amount of programming (e.g., in Excel, MATLAB, Mathematica, LabVIEW).

2. We should strive to create end-user programming tools that make it easy enough for scientists and engineers to do what they need without even knowing that they are programming.

I agree with both of these points. But in the foreseeable future, I think that programming skill will always be positively correlated with creative productivity in many technical fields. Thus, for scientists and engineers who already know some amount of programming, learning more will always provide a competitive advantage over colleagues who are not as adept. We might someday get to a future where programming as we know it will become as obsolete as calculating integrals by hand, but I doubt that is going to happen anytime soon.

Readers' Comments

*FWIW, my research credo (as you may know, Philip) is that we have to go to them, not the other way around. Programming is always a good skill to have, but asking people with immense amounts of domain knowledge (that took years to acquire) to **also** be proficient coders (another skill it takes a lot of time to learn to be competent at) is simply not feasible. Ideally, we should get to a place where the UIs and tools make it so they do not even know they are programming.*

Perhaps put another way, I do not believe there is something special about these domains that requires the ability to do general-purpose programming when many other domains have been having to manipulate data for a much longer time and yet do not require programming skills. We have managed to create special-purpose tools that are narrow enough to be relatively easy to use but broad enough to cover a wide number of problems. Think Excel, for instance. Even professions like financial analysts, at worst, have to contend with SQL. And even that, I would argue, is asking too much.

All that said, perhaps we would agree that, sure, maybe they do ultimately need to program, but there is still a huge mismatch between the programming tools they currently have and what they need. I will agree they need to learn to "program" if you agree we potentially need to change what it means for them to "program." ;)

—Nicholas Murphy

Hi Nick,

I am totally with you. I would love to get to a world where end-user programming tools for scientists are powerful and usable enough that they can do the kinds of sophisticated analyses that they need without even thinking that they are programming. But as it stands, the amount that can be accomplished by existing tools is fairly limited; thus, as a scientist, having some understanding of programming will always give you a comparative advantage vs. your peers who can use only, say, Excel. When programming skills no longer become a comparative advantage in science, that is one sign that the tools have gotten powerful enough. Maybe an analogy is that nowadays, being able to do surface integrals or geometric proofs or draw pretty scatterplots on paper no longer provide a comparative advantage, since computer-based tools make it easy enough to do so.

—Philip Guo

Greg Wilson has been trying to define what part of computing that scientists and engineers need to be productive today. He has been evolving his Software Carpentry workshops (http://software-carpentry.org) to focus on what scientists and engineers need, not everything that computer scientists find valuable.

—Mark Guzdial

Thanks for the pointer, Mark! I am a big fan of Greg's Software Carpentry work and would love to see more efforts to specialize programming curricula for working scientists/engineers.

Many people in the CS Ed community have already done a great job at introducing programming to a variety of audiences — e.g., elementary/middle school kids, high school robotics aficionados, aspiring digital artists — and I would love to see more outreach to the scientific/engineering communities.

—Philip Guo

If I may be so bold as to suggest a fourth reason that it is required:

Programming requires you to break big problems down into their smallest discrete components, and then to solve the big problem by systematically solving those smaller components. This is also exactly the kind of thinking that is required 90% of the time in science and engineering. Once programming has forced you to learn how to think this way, it is far easier to apply it to non-programming problems.*

*in the 90/10 perspiration vs. inspiration breakdown

—Anonymous

I am reading Phil's statement and I'm skeptical about reason 3, which claims one could work 100 times faster using parallel computers. The computer may run 100 times faster, but that will not save me 99% of my time. Usually, computing cycles are not a good measure of productivity. On the other hand, I do not think we need "thousands of hours" to become a programmer. I think it is in the "hundreds of hours." A typical course at a university is less than 40 hours of instruction, and less than 120 hours of assignments. Five programming courses, 5 x 160 = 800 hours, is enough. That is one semester. That is enough to use MATLAB/Maple/ Mathematica/ C++ proficiently.

—Michael Monagan

Philip Guo is a postdoctoral scholar in the Massachussetts Institute of Technology Computer Science and Artificial Intelligence Laboratory.