What Is A Professional Programmer?

by Sarah George

How do people become professional programmers? Many people go the “traditional” path through a computer science or software engineering education and from there into professional programming work.

Others become professional programmers by accident. A person writes a small program to help at work, and their workmates say, “Oh great, you can write programs! You’re our programmer now!”

Other people start out as hobbyists and follow a less traditional path, not always getting a degree, but clearly wanting to be programmers from the start and working actively towards that goal.

I’ve been a hobbyist programmer since I was 6. I wasn’t writing anything amazing back then but I had started writing and soon found it was absorbing most of my time. Since I never really stopped, that gives me 24 years “programming experience” and counting.

At first I was into writing computer games. Later people asked me to write programs for them, and sometimes I even got paid. From this I learned that software is always *for* something. Programs are not self contained worlds of their own. People expect things out of a program that have more to do with Japanese or Geophysics or Engineering (or whatever they’ve got in mind) than with how a computer works. I had to learn something about all those domains in order to write programs for them.

At university it didn’t take long before I was a tutor, and that’s where I found I enjoy teaching, and especially enjoy teaching programming.

While I was at university I got my first “real” job, writing Visual C++ code for a financial database company. In terms of design and theory it was lightweight stuff. But in terms of working with others on a large project I was being thrown in the deep end! They had gigabytes of source code, growing cancerously through the efforts of a dozen developers of wildly differing skill levels.

In spite of my programming skills being well above average there, I learned to settle for being a junior programmer, a little fish in a large pond.

Skipping along a few more jobs and a lot more years, today I am a senior developer in a small research group—a big fish in a little pond. I've had to teach my co-workers a lot about professional programming, because most of them haven't been in industry to get that taste of what large code bases and diverse skill levels do to programs if you aren't using those "professional" skills to keep everyone pointed in the same direction.

There's quite a gap between "being able to program" and being a "professional programmer." It took me 15 years to go from beginner to hotshot programmer, then another 10 years to go from hotshot to professional—and I'm still learning.

Whatever the path we follow, most professional programmers have in common the fact that they learned to *code* first and how to be a professional later.

The Meaning of "Professional"

So what does it *mean* to be a professional programmer? What does it mean to be a professional *anything*? Some definitions simply say to be a professional is "to make money from a skill," but true professionals also have a set of qualities often described as "professionalism." In my opinion, these qualities are: trustworthiness, teamwork, leadership, communication, constant updating of skills, an interest in minimizing risks and accountability. Each of these effect the professional programmer in certain ways.

Trustworthiness The concept of trustworthiness applies in several different ways for programmers. Can you be trusted with a job? To perform a task without someone checking up on you? Can you be trusted to ask for help when you need it?

If you're given clients' data or have signed a non-disclosure agreement, then you are being trusted to respect privacy. You are trusted to check license agreements on third party tools or libraries and to get licenses or permission as required. And like any professional you are trusted to simply do a good job.

Teamwork Will you genuinely cooperate with your team mates? Will you work to mutual advantage and not just your own? Can you trust your team to work with you? Can you do your share of the work and trust your team to do the rest? And can you accept your management (and sometimes even clients) as part of the team, everyone trying to get the same job done?

Leadership Showing leadership means both earning respect from others and knowing what to do with it. Recognize the skills of your team members, and make sure you can offer each person challenges and development without exceeding what they can cope with at a given time.

Leadership involves not always getting to do the "fun" parts of a project yourself (that scary "delegation" word). It also involves not asking anyone to do a task that you wouldn't be willing to do yourself. It's not just the managers and lead programmers who need to show leadership, it's any professional programmer. The best programmers to work with are the ones that know what's going on, not just their little tasks.

Communication Respecting the people you work with, and your clients, enough to *really* listen to them is a critical part of communication. Teamwork can't happen without good communication, nor can accountability.

Communication is critical for helping clients to produce usable specifications and feedback. Will you question whether the specs you are given really will serve the purpose that the client has in mind?

Communication skills help with making meetings timely and effective. A professional's communication is effective and to the point, whether in person, in email, on the phone or in written documents.

Documentation at first seems like a programmer-specific concern until you consider how many people require documentation in a serious project: other programmers need high level, API level and in-code documentation; managers need planning, progress, and bug documentation; lawyers need proof of what was done and when; and users need documentation on how to use the software.

Updating Skills Keeping your skills up to date involves staying aware of what's going on in your industry. What are the current ideas about methodologies like eXtreme Programming? What libraries and tools are out there that might support your project? What are the current refactoring tools? How about standards, file formats and protocols? Are you up to date with Unicode, XML, SQL, and all the other acronyms? Perhaps you're missing out on something if you're not. What platforms are your potential clients using? Should you be learning about cross platform development?

Basically you need to possess a genuine interest in your field, and to read broadly so you know what's out there and which areas to then read deeply about. You also need to accept that even (or should I say "especially") the very best programmers are still learning.

Minimizing Risks Familiarity with best practices, combined with a healthy dose of common sense, will take you a long way towards managing risks. Professional programmers keep track of known bugs or any other change they intend to make. Bugs are risks, and a simple database can prevent you having a product ship with bugs you'd simply forgotten.

Another risk that's often not properly considered is any and all changes to the source code. Source is your livelihood and any change can be a mistake. There's good software out there that will keep track of every revision of your source code and even help merge code that multiple people have changed.

Professional programmers are careful to do enough testing. A software company will generally have testers but the developers need to know how to get the most out of testers and also how to write their own unit and regression tests to make sure every change in behavior is noticed and checked by a human.

Keeping your code simple and well styled is another commonly overlooked way to manage risks. If anyone can look at the code and see right away what it does, you are far less likely to find bugs in it later, and you are less likely to have a junior programmer attempt to change something without understanding it first.

Another risk is the client changing their mind, or more often changing their specifications because they've realized it wasn't what they had in mind. Write your code to be modular and reusable and you won't have any trouble adapting it to changing needs.

Accountability Writing code for others is a *responsibility*. You need to make sure your software is reliable. You need to make sure you and the client truly understand the requirements and specifications. You need to have documentation of your work, all current and past bugs, your progress, any problems, signed-off milestones, and more. You are also required to know about some basic legal issues, like software licensing, the terms of your employment contract, and intellectual property law.

* * *

As you can see, there is a huge gap between "coding" and "professional programming." Most programming courses focus on the coding side of things, and the professional skills tend to be glossed over or not covered at all. I have found myself regularly teaching these skills to new co-workers, which highlighted the need for "professionalism skills training." Teaching my co-workers reminded me how much I enjoy teaching. I decided to teach more people by trying my hand at professional *writing* for a change.

I set up a web site, which is completely independent from my day job. The site is called DevelopingProgrammers.com. It is devoted to teaching people how to develop into professional programmers. Since founding the site, I've been presenting the tools and ideas that I think professionals should know about.

Some of my articles simply refer to other sites of benefit to would-be professionals. I research other articles from scratch: tutorials, guides, and discussions of things professionals should be thinking about, like revision control, documentation, keeping your group pointed in the same direction—and of course, each of the aspects of professionalism that I listed earlier.

These days I consider myself to be a professional programmer, though I am still discovering the depth and breadth of what exactly that means. Perhaps that ongoing exploration of programming and of professionalism is what makes this for me a career and not just a job.

###

About the Author

Sarah George lives in Melbourne, Australia, and holds an honors degree in Computer Science. She has had a range of programming-related jobs, including applied programming, teaching computer science practical and tutorial classes, and is currently working as part of an artificial intelligence research team at Monash University. Her web site, Developing Programmers.com, is aimed at programmers who want to become more professional about their craft.