

developer.***A Web Magazine for Software Developers**

Specialties and Strategies (Part 2)

By Daniel Read

Part 1 of this essay discussed the concept of software development specialties, including how to choose a specialty, how to evaluate the risks and benefits of a given specialty, and how to be aware of the market forces and trends that apply to technologies and specialties. This second part will concentrate more on the “strategies” part of the title of this two-part series.

Advancing Your Knowledge

The subject of monitoring trends (which was discussed in the latter section of Part 1) is directly linked to the subject of advancing one’s knowledge within a given specialty—and within the software engineering discipline as a whole. A truly rich and fruitful specialty will take years to master, and even better, a large base of adopters will lead to continuous innovation, meaning that there is still continuous learning opportunity once mastery is reached—if mastery can ever truly be reached, that is.

How much of a master one becomes is a matter of personal choice. A friend of mine named Trevor, who is also a software engineer, is a student of Shaolin Kung Fu, an ancient Chinese martial arts system. I asked him about the pathway to mastery in this system. (For those not familiar with kung fu, a student progresses through a series of “belts,” with each belt having a different color—white, yellow, green, brown, black, etc. The “belt” is literally just that: a canvas belt that is tied around the waist of the kung fu uniform. Some of the higher belts, like brown and black, have multiple “degrees,” which are sub classifications of progress.)

My friend tells me that, with hard work, attendance at classes several days per week, and discipline outside of class as well, a dedicated student can reach 1st degree black belt in two to three years. This includes mastery of various fighting styles, weapons, and also Tai Chi, which is a more internally focused system of movements and breathing. From the average person’s point of view, becoming a black belt in any martial art is a major achievement, and if you took a survey, I’ll be most people would equate a kung fu black belt with kung fu mastery. But is it mastery?

In the “Shaolin Do” system in which my friend Trevor is studying, there are 10 degrees of black belts. Here’s what Trevor says about the journey of mastery beyond that initial 1st degree black belt: “My instructor, Master Grooms, has been practicing for 25+ years and is a 6th degree black belt, but I think he achieved that more quickly than most, given he practices Martial Arts full time—and there are 4 more degrees beyond where he is. The current Grand Master is 10 years older than Master Grooms and has been practicing since he was a child, so you figure 50 years to get to 10th degree if the stars align just right and your focus is unwavering. There is exactly ONE 10th degree Shaolin Grand Master in the world—that’s it.”

I don't know about you, but to me, that's pretty intense. Three years to reach the first "level" of mastery, and then a lifetime to refine that mastery further and further. It certainly offers a different perspective on the idea of "mastery." This kung fu analogy, however, probably does not hold up when comparing it to any one of the fairly narrow software development specialties we're discussing here. Since these kung fu masters are mastering a whole range of systems and techniques, it would probably be better to use the kung fu mastery analogy to compare with the discipline of software engineering as a whole, which encompasses a whole range of specialties and sub-specialties.

Regardless, my point is this: once we have a basic competency in software development (and I'll bet a kung fu Grand Master would consider the esteemed 1st degree black belt as merely "basic competency"—it's all relative), if we want to attain "mastery," and the higher pay, visibility, and satisfaction that comes with it, then we cannot stop our knowledge seeking. We must continually dig deeper into the discipline, yes, but we must also study *outside* of the discipline into non-technology-related domains, in order to introduce fresh ideas and more profound understanding. Personally, I feel like my journey is still just beginning.

Before moving on, I'd like to touch further on the concept of non-technical knowledge and growth. Gerald Weinberg is the one who really turned me on to this concept. Through his books *Understanding the Professional Programmer* and *Becoming a Technical Leader*, and through his Problem Solving Leadership workshop, which I was fortunate enough to attend, I have learned that the technical challenges we face as developers are the least of our concerns.

At a certain point, the technical stuff becomes easy—you start to see the same patterns and principles over and over again. That's a bit of an overstatement, since if there were not continual technical challenges we would likely get bored, but it's all the "human factors" stuff that's really hard—and what's hardest is improving the one human each of us has to deal with every day: ourselves.

Protecting Your Specialty

If you are good at what you do, then you are constantly going to be beset by people trying to get you to do other things that you don't necessarily want to do. Most of the time, you can take this as a compliment, because it means that someone has recognized your talents and/or work ethic. However, knowing when and how to say "No" is important if you want to control your own career destiny.

Determining when to say "No" when someone asks you to do something can be tricky. One key factor is the duration of the task. If someone is asking for your help on a relatively minor task that you can knock out pretty quickly, then you might want to accept the task. That's just part of being a team player: if something needs doing, then it does not reflect well upon you to say "No" because you feel you are above that sort of task. Let's look at an example.

Mary is a software engineer in a large company that employs a high number of technology professionals. Mary's specialty is writing object oriented C++ programs, and she would like to continue doing that for the foreseeable future. Mary's code for the latest release of the software is pretty much done except for a few debugging tasks that are arising out of the QA process. The team is committed to pushing this release to production in a week. Mary's boss comes to her and says, "Mary, I'm in a bind. Harry is on leave for the next month, and as you know, Harry is in charge of writing and generating the installation programs for the system. You are done with your code, and I need your help getting the installation program done and tested in time for the release. You're the only one whose schedule is not already full for the next week, and I know you're more than capable of doing a great job."

What should Mary do? In my opinion, she should say "Yes" to this task, since it is a one time thing, it should take a relatively short time to complete, and her team needs her to come through in a pinch. If Mary says "No," then her boss would be right to be disappointed, and, fair or not, might even feel that Mary is being selfish. What if the situation were a little bit different, however?

What if Mary's boss comes to her and instead says, "Mary, I'm in a bind. Frank, our report developer, just quit the company. Your coding on this system is done, and as you know this is pretty much the final release of the system for awhile. Frank was scheduled to design and build forty reports over the next three months. I need someone to fill in for Frank since I don't have time to bring on a new report developer, and the company has a hiring freeze right now anyway. I really need you to take Frank's place for the next few months and develop these reports. Will you do it?"

Here the correct course is a little more gray. We are no longer talking about a simple matter of Mary taking on a quick task, and then getting back to work on her main specialty, C++ coding. Let's assume that Mary recognizes that she does not really *want* to develop forty reports over the next few months. But can she say "No"? That depends a lot on Mary: how she feels about protecting her specialty as a C++ developer, how she feels about her future with this company, and how she feels about the demand for her skills on the open market.

Maybe she has been wanting to scale back her responsibilities and take it easy for awhile, and feels that this report building gig would be okay for a few months. Maybe Mary knows that if she says "No" to her boss that there is a high likelihood she could be laid off, since the company is going through a rough period, and there is not a ton of C++ work waiting to be done. Maybe she really wants to stay with the company long term, and the sacrifice of doing reports for a few months is a small one compared to her career with the company. Maybe she knows she has an inflated salary relative to what the market is paying right now for C++ developers and is not willing to risk losing that. These are all valid reasons that Mary might say "Yes" to her boss's request. I certainly would not judge her for saying "Yes," since these kinds of decisions are very personal, and there really is no right or wrong answer.

Mary, however, might have different ideas about her specialty as a C++ developer. If she is confident in her skills and confident in her ability to find an equivalent or better new job if it came to that, then she might decide to politely decline her boss's request. The danger that

Mary is wary of (and rightly so) is that she might get stuck as a report developer indefinitely, which might allow her C++ skills to get rusty, might keep her from working on new skills and technologies within her C++ specialty, and might lead to her company's forgetting entirely that her specialty is C++ and not report building. What happens if her boss is promoted, transferred, or fired two weeks after she agrees to take the reporting gig? The new boss will see her not as a C++ developer but as a report developer. Also, if she gets stuck writing reports for several months, then she will be forced to reflect that on her resume, which will look strange to potential future employers. She should also consider whether there is a hidden message in her boss's request—could her employers be telling her something about their feelings about her C++ coding skills?

Making a Change

You may have already recognized that this situation of Mary's might be a very positive one if she was seeking to change specialties to report development. If that were the case, she would of course jump on the opportunity to take a report developer's position. When you are seeking to change specialties, for whatever reason, then these are exactly the kinds of opportunities you want to look for, and if possible create.

There are various ways to create opportunities like these. My favorite method is to just work on the side doing what it is you want to be doing, and then present it to your employer after it's already done. Use an unfamiliar technology or technique to solve a problem that your employer did not even know she had. Assuming that you've solved this problem on your own time, and without neglecting your assigned work, a good employer will appreciate your initiative and recognize your abilities to use the new technology or technique. You can explicitly come out and tell your employer, "This new thing is what I really would like to be doing, and I was hoping that there would be opportunities here for me to do it." You have to be in the right kind of situation with the right kind of employer for that strategy to work, however. It might be that in order to change specialties, you will have to change employers. Even in that case, though, I don't think it would be disingenuous to put that new technology or technique on your resume.

The Pull Upward

Many developers, especially good ones, will eventually feel the pull toward management. This is the ultimate specialty protection dilemma, because when a developer is promoted to management, coding opportunities tend to cease to exist. I myself have had to resist this pull many times, and in my current position, which is a quasi-management position, I decided to let the management pull win a little. It's been about a year since I've had an opportunity to write any real code. Taking this position was a conscious decision, and at the time that I made it, I knew it was only going to be for a limited time. As I write this, some major changes out of my control have occurred at my current employer, and I am seeking a new position, where I will hopefully be able to get back into the trenches building software again. However, I don't regret my recent decision to take a management position for awhile. I learned a lot about myself and the process of leading a team to build great software. Watching other

people write code and work under deadlines has taught me more about myself as a developer.

Getting promoted from the trenches to the command tent can be a little scary and a little sad—scary because you are suddenly on unfamiliar territory, dealing more with people than with technologies, and sad because you are leaving behind an activity you love. People on your team that you formerly worked side-by-side with will view you a little differently, and won't share their thoughts and feelings with you so readily as they used to. Politics and meetings will suddenly become a big part of your life. You will often feel caught in the middle between your obligations to the people working for you and your obligations to the company and your bosses. That said, management can be a rewarding job.

If you do not wish to make a move to management at this point in your career, be prepared to resist the pull and say "No."

Cultivating the Right Kinds of Generalization

Given the diversity of programming languages, platforms, techniques, etc. it is inevitable that a software developer will master a limited subset of the possible software engineering specialties. There is just too much diversity for anyone to be an expert in everything. However, too much specialization is a bad thing. We must also be generalists. A software developer who is too focused on the mechanics of writing code or making a certain technology work will find himself limited career-wise—not to mention, without certain generalist skills, you simply will not be able to develop software that is acceptable to very many people besides yourself.

This discussion is not the right place to delve into the details of proper software engineering generalization, but below is a list of knowledge areas and skills we would all do well to learn something about:

- Requirements gathering
- Design (this will take many different forms, depending on the technology(ies) in use)
- Architecture
- Quality assurance and testing
- Deployment and implementation
- Change control and configuration management
- Algorithms
- Documentation
- Usability engineering
- Transaction control
- Methodologies and process
- Metrics
- Leadership
- Diplomacy
- Conflict resolution and management

- Code and artifact reviewing
- Risk management
- Marketing

As previously mentioned, there is also a lot to be gained from focusing on learning outside the field of software engineering. I would say that just about any discipline or subject has something to teach that will be useful to any of us as software developers. Here's a short list of a few subject areas that are worth attention (not that I can pretend to have studied these areas extensively myself—all of this advice is directed as much to myself as to anyone else reading these words):

- Sociology
- Anthropology
- Psychology
- Mathematics
- History
- Philosophy
- Religion
- Engineering
- Medicine

I have also gained insight from reading the early influential works of the software field. It's amazing to see how software developers thirty years ago were dealing with the same issues we deal with today—the acronyms are just different. Sadly, much of this material is out of print, but seeking out a few books such as *Classics in Software Engineering* (edited by Edward Yourdon) and *Software State-of-the-Art: Selected Papers* (edited by Tom DeMarco and Timothy Lister) might be well worth your trouble.

Conclusion

The point of this whole discussion has not been to tell you what to do or how to feel. Career management is a very personal topic, and there are no right or wrong directions or decisions—there is only what is better or worse for you. My intention is merely to point out that conscious thought and strategizing are required in order to acquire, maintain, and build on a software development specialty, and thereby to build a career as a professional software developer. As I said near the beginning of this two part essay, the only one who can manage your career is you.

###

Daniel Read is editor and publisher of the **developer.*** web magazine. He lives in Atlanta, GA, where he works as a software architect. He is currently at work on a book about software development crafted for a business audience.