Editor in Chief: **Diomidis Spinellis**
Athens University of Economics
and Business, dds@computer.org

Being a Software Developer

Diomidis Spinellis

BEING A PROFESSIONAL software developer is tough." This is what I told a manager lamenting the large number of university graduate job applicants who couldn't pass his company's coding tests. Programmers develop the most complex human artifacts by manipulating symbols that are defined through layers upon layers of abstraction. The resulting instructions are then processed at a rate of billions per second—often in a nondeterministic fashion—on atom-scale processors, which become 10 times more powerful every five years. A computer science, informatics, or related degree can only scratch the surface of the required knowledge and skills. A specialized software engineering degree can go into greater depth but still can't offer all the required expertise and experience. Consider that it takes many years before a medical- or flying-school graduate is allowed to perform open-heart surgery or fly an Airbus A380, respectively.

Because work on production code quickly becomes very demanding, if you want to be a professional developer, you'll need to continually invest substantial time to acquire highly specialized knowledge and develop diverse skills. A university can kindle your passion and provide incentives to expand your horizons, and your employer may support specialized training. But in the end, becoming a professional software developer is your decision and responsibility.

Knowledge

In the original version of Bloom's cognitive-domain taxonomy of learning, the first element was knowledge.[1] A university degree will give you significant theoretical knowledge that's required for writing software. This includes choosing appropriate data structures and algorithms, understanding how the computer's architecture and OS affect performance, using appropriate programming-language features, and applying software engineering methods. Many curricula typically will also cover context-specific specialized knowledge, such as human–computer interaction, computer graphics, information security and management, networking, and intelligent systems. What a university program typically can't offer is specialized domain know-how (e.g., computing applications in civil or aeronautical engineering) and comprehensive knowledge of software construction tools.

As a professional developer, you need to advance from the sandbox development tools that are often used for teaching programming and become productive in a full-featured IDE and a powerful general-purpose code editor. Examples of such tools include Eclipse, Visual Studio, Vim, Atom, and Emacs.

You'll also need in-depth knowledge of a general-purpose programming language, such as Java or Go, and an application development framework or API, such as .NET. By the time you've become accustomed with all of a language's features, you'll have gathered enough confidence to be productive in everyday tasks. Similarly, being somewhat familiar with all the areas covered by a framework will allow you to find and choose its appropriate features for each situation.

Finally, you'll also need to be thoroughly acquainted with a wide set of software construction tools. These include tools for build automation, configuration management, debugging, testing, static analysis, continuous integration, and package management. You should be able to not only configure and run them to cover specific requirements but also grasp the principles behind their operation. How does Git store revisions? What's the transitive order of build or package dependencies? Why do false positives and false negatives arise in static analysis? What's a data breakpoint? Given the variety of ad hoc software construction tasks you'll be facing regularly, you'll also need mastery of some general-purpose toolsets, such as the Unix command-line tools or Python and its modules.

Cognitive Skills

In contrast with knowledge, most of the cognitive skills you'll require as a software developer can only be indirectly taught at a university. Some of them, such as those related to critical thinking, are even honed in much earlier life stages. Still, you can always improve your weak areas through practice, introspection, and training. Unsurprisingly, a developer's key cognitive skills include all the remaining elements in Bloom's taxonomy: application, comprehension, analysis, synthesis, and evaluation.[1] Here are some examples.

Applying your theoretical knowledge means generalizing specific software requirements (say, the association between customers and products) into the corresponding concepts (here, modeling a many-to-many relationship). It might also mean applying quantitative reasoning to evaluate things such as software reliability, the user experience, or system performance. (How will a 5 percent increase in a lock's contention affect transaction latency?) Other common applications of your hard-earned theoretical knowledge involve choosing appropriate abstractions (e.g., the use of implementation inheritance, interface inheritance, or parametric polymorphism); taming complexity by structuring code; and applying existing methods, tools, APIs, and algorithms to solve specific problems.

Regarding *comprehension*, you must be able to interpret existing code sequences, extend them to match new requirements, and refactor them to reduce technical debt. You must convert specifications into code, summarize code as concise comments, and explain it to your colleagues. You must also provide lucid narratives and examples of specified processing and express them both in clear writing and as integration and unit tests.

In your software development work, you'll require *analytical* skills to infer possible causes of bugs and performance issues, to subdivide a complex system into more manageable elements, to prioritize requirements, and to classify mind-numbing special cases into broader categories. Analytical skills also come in handy when you're comparing competing software or user interface designs for implementing some functionality, in order to select the most appropriate one.

Hand-in-hand with analytical skills, which help you break bigger problems into smaller pieces, go *synthesis* skills, which combine existing fragments into more valuable aggregates. A prime example here is the ability to design a large complex system as a composition of several existing and bespoke software components. Other applications of

CONTACT US

AUTHORS

For detailed information on submitting articles, access www.computer.org/software/author.htm.

LETTERS TO THE EDITOR

Send letters to

Editor, *IEEE Software*
10662 Los Vaqueros Circle
Los Alamitos, CA 90720
software@computer.org

Please provide an email address or daytime phone number with your letter.

ON THE WEB

www.computer.org/software

SUBSCRIBE

www.computer.org/subscribe

SUBSCRIPTION CHANGE OF ADDRESS

address.change@ieee.org.
Please specify *IEEE Software.*

MEMBERSHIP CHANGE OF ADDRESS

member.services@ieee.org.

MISSING OR DAMAGED COPIES

help@computer.org.

REPRINTS OF ARTICLES

For price information or to order reprints, email software@computer.org or fax +1 714 821 4010.

REPRINT PERMISSION

To obtain permission to reprint an article, contact the Intellectual Property Rights Office at copyrights@ieee.org.

EDITORIAL STAFF

Senior Editor/Magazine Contact: Dennis Taylor
Publications Coordinator: software@computer.org
Multimedia Editor: Erica Hardison
Cover Artist: Andrew Baker
Director, Products & Services: Evan Butterfield
Publisher: Robin Baldwin
Manager, Editorial Content: Brian Brannon
Senior Advertising Coordinator: Debbie Sims, dsims@computer.org

synthesis include low-level code construction by utilizing language features and APIs, deriving a hypothesis regarding an underlying fault from its diverse manifestations, planning a Scrum iteration, or coming up with a new algorithm.

Finally come *evaluation* skills. As a developer, you'll need to perform code reviews; identify technical debt; assess designs, test methods, and processes; and recommend the most appropriate solutions among a wide variety of technical alternatives.

Interpersonal Skills

Software development relies enormously on interpersonal skills because in most cases it combines extensive teamwork with considerable individual leeway and responsibility. These skills are also rarely formally taught at a university.

First in this area come skills associated with collaboration. To work effectively as part of a team, within your organization or with your customers, you need to be able to receive and provide constructive feedback, to show appreciation for the work of others, to actively listen to them, and to collaborate remotely. Often you'll receive criticism—sometimes brutally honest—through code reviews, testing, and customer feedback. You must be adept in using this input to improve your designs and your code without taking it personally. You also need to be able to work smoothly within an organization's hierarchy: to be managed and to manage others.

Then comes the work ethic. This involves professionalism, respectfulness, dependability, a positive attitude, and adherence to the workplace's and the software community's etiquette. You must be able to provide honest, accurate schedule estimates and dependably deliver on them. You must appreciate and respect your organization's conventions regarding code and development processes as well as common work ownership. More broadly, you must contribute back to the professional and scientific communities through Q&A forums, open source projects, conference presentations, and publications.

As a developer you'll often be responsible for considerable aspects of your organization's software. This requires self-confidence to accept those responsibilities, persistence to deliver what's required from you, and thoroughness and perfectionism to maintain high quality. Remember: a misplaced bracket can break an entire build; a missing error check can destroy an entire organization.

Being a professional in a rapidly evolving field means you need to continually invest in learning new skills, technologies, practices, and tools. Be on the lookout for them by reading professional magazines (including this one), attending conferences, studying classic books as well as new ones, and examining other people's code (open source software makes excellent such material). Look for aspects of your and your organization's work that can be improved, and be ready to learn and experiment with new methods to address the shortcomings.

Cruelly, complex skills can quickly deteriorate. This is why even private pilots are required to perform a minimum of three landings within 90 days and pass a biennial flight review to keep their license current. For you as a programmer, this means continuing to program—at work or as a hobby—even as you take on more management responsibilities.

Building and maintaining all this knowledge and these diverse skills seems like a tall order. This is the entry price for a hugely rewarding profession. A profession through which you can improve the lives of countless individuals, the effectiveness of businesses, and the functioning of governments. A profession that lets you use your brain and a keyboard to make our world a better place.

Reference

1. B.S. Bloom et al., *Taxonomy of Educational Objectives: The Classification of Educational Goals. Handbook I: Cognitive Domain*, David McKay, 1956.