Programming Will Never Be Obsolete

Andrew Mellinger, SEI

Abstract. We live in world that will always be full of problems. Changing conditions and advances in science and current solutions are constantly providing even more opportunities daily. While these areas may share similarities to previous problems, the essential fact that they have not been solved means that creativity is required to provide a new solution. It is this need for creativity that prohibits machine and algorithms from dealing with this issue and that we will need a programmer to translate these solutions into executable form.

Programming Is and Always Will Be Important

We have all heard the argument that programming will become obsolete. Notions like "it is a dead end career" or "salaries will drop" are constantly plaguing the viability of the field. A quick web or periodical search will return articles on the topic from at least as early as 1984, and there are new ones being posted every day. They range from scholarly articles such as, "Can fifth-generation software replace fallible programmers?" to modern blog posts that cut to the chase, "Is Programming Really as Future Proof a Profession as People Think?" [1] [2].

The issue is raised for a variety of reasons, some of which are honest and some are disingenuous. I prefer to focus on the genuine concerns of developers, technologists, and academics that the end of programming and their careers will be brought on by automating programming tasks or the end of a particular technology on which they depend. I will ignore disreputable claims that the problem can be solved by adopting a certain vendor's technologies or getting particular platform certifications.

Often, people will see a decline in a particular technology or method and will prophesize the fall of programming generally, rather than as it pertains to the specific technology. The need for programming may decline for programmers near the end of a specific technology's lifecycle, but the general technological challenges are moving targets, and therefore, we will always have new problems.

When discussing programming, some people are referring to the act of typing in the code, and some mean the entire software development lifecycle. This article includes all aspects of development and will use development (developer) and programming (programmer) synonymously. Programming, development, and engineering are highly related activities but focus on different dimensions of the overall software production process. The difference between these high level activities is the degree to which they directly interact with code. Programming is the activity that is closest to the code, while engineering is generally the farthest. Programming is where the developer picks and chooses from the available technologies, patterns, accumulated practices, techniques and their experience to "best" satisfy the complex interaction of requirements.

It is this fundamental interaction with the code that differentiates the actual act of programming from other activities. Programming should be not conflated with the physical act of typing, but equated with the "last mile" of actually coding, or expressing the intent in an executable language. Some would argue that this is simply a translation process, but for anyone who has worked on a project of substantial size, it is much more. In simple natural language translation the input and outputs of both are at the same semantic level. For example, if I am translating "My hovercraft is full of eels" from English to Swedish, I am trying to say the exact same thing in both languages. In programming there is a change of semantic level. For example, the requirement may be to "support undo" which implies a variety of user interface interaction points, interactive behaviors, and changes to storage semantics. One may argue that undo is a complicated concept and should not be handed to a "programmer" but in practice projects frequently hand problems of this complexity to a developer, or the person who is touching the code. Modern frameworks have a lot of infrastructure to support complex patterns like undo, but there are still a wide variety of decisions to be made by the developer with regards to the domain specifics.

Eras in Technology

Technologies rise and fall in popularity, and while they drive business growth they also require a tremendous amount of programming. New technologies arrive with a bang and drive the economy for some period of time through tooling, employment, and products. These periods, or "eras," vary in size, length, and overall impact. Eras overlap with those of other technologies such as different languages, software platforms, hardware architectures, peripherals, and development methods that draw an incredible amount of innovation. Consider the iPhone, which was introduced in 2007 and opened up new economic and technological markets. At that time there was a huge demand for Objective-C/Cocoa programmers and people who understood the special nuances of mobile device interaction and their interfaces. The iPhone impact had a ripple effect through the tech industry and ushered in Android technology, which introduced an increased demand for Android/Java programming. Then the tablet arrived and created a tablet/phone hybrid tsunami.

During each technological era we see cutting edge technologies move from the inventors and innovators to early adopters and eventually adoption by the masses. Most successful eras possess similar qualities such as a wealth of new ideas, financial investment, fierce competition, and general uncertainty. How does a developer live through this cycle? We are bombarded by a wealth of new technologies touted by vendors, researchers and volunteer communities. Which do we choose to learn? What do we follow? It is impossible to

review, much less understand every language, framework, tool or platform that arrives and we need to choose some that keep us fresh and might help our current job. At some point in each technological era, the cutting edge becomes not so sharp and the leaders are identified. This is the time period that makes it easier to choose which technologies you should learn and adopt. Eventually, the era progresses to the point where the technology area enters the mainstream. This is when we typically see the publication of books on the subject, and finally the emergence of the "standard" technologies, protocols, or methods.

Over time these technologies become commonplace when point-and-click tools, or off the shelf packages that are suitable for a vast majority of the instances. As is a programmer's nature, when we see something that is "routine" we write a script or app or framework to do it faster, cheaper, and better. This is when you will see a decrease in the need for the specialized skills and training of a programmer. However, this will also usher in its own set of doomsayers and charlatans. What is becoming "obsolete" or in less demand is the need for a particular set of skills, not for technology problem solvers.

The Programmer's Role

When I interview people for programming positions, I divide them into two categories: programmers who focus on a particular technology and programmers who focus on the underlying principles of technology. A programmer that advertises themselves as an "insert-favorite-technology-here developer" instead of as a "software developer" is more likely to learn one or two skills the market needs and work exclusively within those roles. I refer to this type of programmer as a "technician" as opposed to a "general purpose developer." The technicians are often the people who argue that their favorite technology is the solution to all of your problems. While they may be masters at that technology (or a handful of them), their fate is inevitably tied to it. Do not get me wrong, these can be tremendously creative, talented, and smart people, but they have a very limited focus. When that technology declines they will find themselves having difficulty finding work and will blame it on the fact that "programming is dying" when in reality they have not stayed relevant.

General purpose developers are not tied to a technology, they have tied to technology. They get bored working with just one technology, which is good. This drives them to attempt to automate things and make technology cheaper, faster, and better. These developers are ready to move to new languages or platforms as they become available because they are not focused on one technology. Development requires decision making and creativity, which are two things we cannot automate. Granted, general developers may become focused (sometimes obsessively so) on a technology for a while, but eventually find the need to tie their work to a general computing and technology problem. The ability for programming generalists to be creative and apply fundamental programming principles to build new technologies is the cornerstone that continues to make them cutting-edge and essential to business growth.

Fundamentally, computers are good at doing what we tell them to do. This means that someone must understand what we want them to do in the first place. A software developer's fundamental job is to take knowledge and make it "executable" or "actionable." The job also requires discovery of this knowledge through requirements definition, usability studies, domain analysis and prototyping. Software architecture, design, and coding all require a significant amount of analysis, reasoning, and decision making. Consider that so many companies want their developers to provide "revolutionary" products, and we can see that creativity will be a requirement for years to come.

Essentials of Programming

We will not run out of problems to solve. Whether they are core research problems or applying some set of solutions to a particular job, we need to look at what the essential qualities of programming are and why they will persevere. Even if we create a solution to a problem, the solution itself is likely to create new problems.

In "No Silver Bullet—Essence and Accident," Fred Brooks argues that software development is so challenging that it will require human intellect for a long time due to four fundamental qualities: complexity, conformity, changeability and invisibility [3]. These qualities have not changed since he wrote the article over 25 year ago, and do not seem likely to change. It is these same qualities that we are trying to use technology to solve, but it is technology that keeps moving the problem ahead of our solutions.

On the implementation side alone, as we continue to discover and learn more, we will always need someone to translate that knowledge from the domain into something executable. We will always need someone to fill that gap as there will always been that point where a person can make an executable representation but where it is not routine enough to automate. We will always be encountering new problems and the sheer nature that they are new problems means they have not been solved. Certainly, many problems in that class may have been solved by many long nights by developers, but not the general problem itself. Even when reusable patterns exist such as a framework, technicians will be required to encode a specific instance such as a particular website or cloud instance for that problem.

When we take all of this into consideration, programming as a creative work will cease to be needed when we have automated all other creative knowledge work. We are more likely to make lawyers, insurance salesman, or politicians obsolete before programmers. One could argue that in the very far future once we have discovered everything and can finally automate the very last thing, that last job will be for a programmer.

Being a Developer in the Future

So what will programming be like in the future? At its core, it will be like it is now. Developers will work to understand the domain, do general problem solving and knowledge creation and then instruct machines on how to execute these solutions. They will need all the skills of the general developer and some understanding of their domain. And they will need to be able to learn and adapt. Marc Andreessen argued in "Why Software Is Eating the World" that as more and more things include a software component, general software developers will always have new problems to tackle [4].

What are the next possible technology areas? A quick glance at the Gartner Hype Cycle can help us prepare [5]. Mobile and cloud technologies are well underway, but that space is very broad and deep with tremendous needs of usability, security, and big data. We have barely scratched the surface with autonomy, ubiquitous computing and the broad application of 3D printing; much less the ones further out such as nanotechnology or biotechnology. Some of these are not computing problems, at least how we know it now, but will certainly require "programming" of some sort. One can peruse modern science fiction to see how a programmer's world might be different in the years to come.

We live in a world that will always be full of problems. Changing conditions and advances in science and current solutions are constantly providing even more opportunities daily. While these areas may share similarities to previous problems, the essential fact that they have not been solved means that creativity is required to provide a new solution. It is this need for creativity that prohibits machine and algorithms from dealing with this issue and that we will need a programmer to translate these solutions into executable form. On the other hand, the specific technologies will change as we routinize these tasks and climb the abstraction ladder. Because of this, specific programming and programmers may become obsolete, but new problems will always require new solutions and general programmers to implement them.

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

DM-0000806

ABOUT THE AUTHOR

Andrew Mellinger is a member of the technical staff at the SEI. His passion for computing started at age 12 when he wrote his first commercial piece of software for the company where his father worked. He currently focuses on data intensive scalable computing, security informatics, cloud computing, and adaptive and heterogeneous architectures.

Software Engineering Institute
Carnegie Mellon University
4500 Fifth Avenue
Pittsburgh, PA 15213-2612
Phone: 412-268-5800
Toll-free: 1-888-201-4479
<www.sei.cmu.edu>

REFERENCES

1. Philips, R. Can fifth-generation software replace fallible programmers? Computerworld, v 18, n 29, 1D/27-30, 16 July 1984
2. Perry, Jon; Kupper, Ted Is Programming Really as Future Proof a Profession as People Think? Accessed November 2013 http://declineofscarcity.com/?p=2557
3. Brooks, Frederick P. No Silver Bullet: Essence and Accidents of Software Engineering. IEEE Computer, vol. 20, pp 10-19, 1987 <http://www.gartner.com/technology/research/methodologies/hype-cycle.jsp>
4. Andreessen, Marc. Why Software Is Eating The World August 2011. <http://online.wsj.com/news/articles/SB10001424053111903480904576512250915629460>
5. Gartner, Inc. Last accessed November 2013