DOI:10.1145/3469115

Brett A. Becker

▸ Mark Guzdial, Column Editor

Education

What Does Saying That 'Programming Is Hard' Really Say, and About Whom?

Shifting the focus from the perceived difficulty of learning programming to making programming more universally accessible.

The commonly held belief that programming is inherently hard lacks sufficient evidence. Stating this belief can send influential messages that can have serious unintended consequences including inequitable practices.[4] Further, this position is most often based on incomplete knowledge of the world's learners. More studies need to include a greater diversity of all kinds including but not limited to ability, ethnicity, geographic region, gender identity, native language, race, and socioeconomic background.

Language is a powerful tool. Stating that programming is hard should raise several questions but rarely does. Why does it seem routinely acceptable—arguably fashionable—to make such a general and definitive statement? Why are these statements often not accompanied by supporting evidence? What is the empirical evidence that programming, broadly speaking, is inherently hard, or harder than possible analogs such as calculus in mathematics? Even if that evidence exists, what does it mean in practice? In what contexts does it hold? To whom does it, and does it not, apply?

Computer science has a reputation[3] and this conversation is part of that. Is programming inherently hard? Although worthy of discussion, this Viewpoint is not concerned with explicitly answering this question. It is concerned with *statements* such as "programming is hard" and particularly the direct and indirect *messages* such statements can convey. It explores who says this, why and how it is said, and what the ramifications are. Consider the statement "computer programming could be made easier."[7] Although this implies that programming is possibly more difficult than it needs to be, it clearly sends a different message than "programming is hard." This exemplifies how two fairly similar statements can convey very different messages and likely have different effects.

Who Says Programming Is Hard?

The belief that programming is hard seems to be widespread among teachers and researchers.[4] Academic papers frequently state that programming is hard anecdotally, as if just stating the obvious. Yet it is rarely discussed outside of motivating and justifying research. Although this approach is rarely challenged, when it is, the stakes are high. One critique points out this stance can lead to uncritical teaching practices, poor student outcomes, and may impact negatively on diversity and equity.[4] That work suggests the expectations of educators are unrealistic—not that programming is too hard.

The message of difficulty is also carried through more everyday mechanisms. It can be unknowingly or inadvertently perpetuated through our teaching habits, textbook language, terminology, the defensive climates in our classrooms,[3] tools, and programming languages themselves. A case in point is programming error messages that, across almost all languages, are notorious for causing confusion, frustration, and intimidation, and have been described as mysterious and inscrutable.[1]

The belief that programming is hard is not confined to academia. The concept of the "10x developer"—the elusive developer that is 10 times more productive than others—serves to communicate that programming is hard and few can be good at it. Even professionals have referred to programming as a black art,[8] a view that persists to present day for some. Hollywood typecasts embodying the hacker stereotype, staring at screens while 1s and 0s quickly stream by, present programming as a mystical, supernatural ability. It is possible that such portrayals have negative side effects in addition to their entertainment value.

Known Difficulties and Overlooked Successes

It is more accurate to say that certain aspects of programming are difficult or more challenging than others. This considerably dilutes the notion that programming is innately hard, as some aspects of many endeavors are more difficult than others. More pointed statements are also less likely to inflict collateral damage on general audiences and are less prone to misuse. Aspects of programming that are accepted to be challenging include knowledge transfer issues—including negative transfer—and developing a notional machine, among others.[3] Programming has several candidate threshold concepts[5] but so do aspects of many disciplines.

Programming is also the subject of many misconceptions.[5] This might be because it is hard. On the other hand, saying it is hard might be a convenient way of explaining these misconceptions. Other (and generally older) disciplines also have challenges with misconceptions. Many physics students struggle with the fundamentals of mechanics such as force vs. acceleration, speed vs. velocity, weight vs. mass, and the concept of inertia. Such comparisons are not that indirect. These well-established concepts underpin a more formulaic approach in much practice, similar to how the conceptual underpinnings of programming are expressed ultimately in code. Engineering also can suffer from a hard image. The notion that mathematics is hard already exists in many school systems and is often echoed by parents and other stakeholders, leading to negative implications including working against broadening participation. However, computing casts a very wide net in modern society; it is intertwined with much of daily life and influences chances of success in many ways. Similar could be said for mathematics but computing may seem more visible and tangible to many. It is also a very attractive and in demand career path for tomorrow's graduates.

The belief that programming is hard seems to be widespread among teachers and researchers.

There are examples that support programming not being hard. Success has been found in pair programming, peer instruction, worked examples, games, and contextualized approaches such as media computation.[3] These are often backed by empirical evidence[5] but are frequently overlooked when convenient. Block-based programming has become extremely successful, particularly with younger children. New modalities also demonstrate that computing is relatively young and rapidly changing. This pace of change itself may be one of the leading factors contributing to the perceived difficulty of programming.

Hard For Whom?

Given how many people are affected by computing technologies, another problem with statements that programming is hard is that most research and media reports are based on very narrow samples of Earth's population—largely from American, Commonwealth, and European contexts. There are entire countries, nearly entire continents, and countless groups whose experiences have not been rigorously studied and contrasted with others. Even in more frequently researched locales, our current views are not representative of many. This is rarely acknowledged or acted upon. Many sub-populations in terms of all manners of diversity and identity including ability, culture and ethnicity, geography, gender, native language, race, socioeconomic background, and many others are still underrepresented, everywhere.

Additionally, most data and experience currently come from computing students, yet computing is quickly becoming a mainstream discipline embedded in school curricula and for an increasing range of academic disciplines in higher education.[5] Declaring programming to be hard for relatively well-resourced Western computing students paints a bleak view for others. Even if programming was found to be uniquely hard for these students, this finding might only hold for the very limited, biased samples that have contributed to our current knowledge. Carelessly attempting to generalize such a finding would serve to shut the door on those who have not yet had their experiences counted. In effect, such practice is already happening when the message that programming is hard is perpetuated with little or no context.

Ramifications

There are several examples where explanations for observations in computing education have unintended negative consequences. For instance, to explain struggling students sitting next to high achievers, the "Geek Gene" hypothesis proved convenient. This states that programming is an innate ability. In other words, one generally cannot learn to be a great programmer; one is or is not, and most are not. There is evidence that computer scientists believe that innate ability is more important in computing than in other disciplines and this is known to be a barrier to broadening participation.[3] Although the Geek Gene hypothesis has met resistance[3,8] damage has already been done and might continue.

One need not look far for other contemporary examples of unintended messages having undesired effects. It was recently shown that competitive enrollment policies for university-level computing majors have a negative impact on student sense of belonging, self-efficacy, and perception of department.[6] Of course, these were not intended outcomes. Competitive enrollments were largely a response to growing student numbers and demand that could not be met. Nonetheless, this mechanism sent an unintended signal to students, resulting in undesired negative consequences. It is likely the perpetuation of the message that programming is hard has similar effects.

A recent series of NSF workshops revealed one of the most-heard comments made by non-computing educators when discussing computing curricula was "stop making computing/programming look scary."[2] Is that really the image that we intend to portray or is it just a byproduct of computing culture? If educators think that programming is scary, how can we expect students to think any differently? These messages may already have resulted in countless students abandoning computing, or not considering it in the first place. We will never know. It is likely that untold damage has already occurred and continues to accumulate.

The Future

The relatively short history of programming is filled with constant change, and what is taught can change often.[3] Character-based high-level programming has, overall, led the battle for adoption particularly at university and in industry, but there are many examples of programming that do not fit this mold. From Logo and Hypertalk, to prototype spoken-language programming languages, to block-based programming, and domain- and task-specific programming, what exactly constitutes programming can depend on who is asked and when they are asked. Today, low-code and no-code are emerging programming modalities, another sign that what constitutes programming is in constant flux. The number of programmers in non-computing contexts is also rapidly increasing.[1] Surely even if programming was deemed hard yesterday, that does not mean it will be tomorrow.

How can we change programming's notorious reputation? The answer is likely multifaceted and includes being aware of the true effects of the beliefs we hold and the messages we send. In addition, these should be based on evidence, informed by an appropriate diversity of people. We should have realistic expectations of students and focus on what we know is successful both in practice and in our messaging, including examining the intent of statements on the matter.

Conclusion

The notion that "programming is hard" is frequently reinforced in our classrooms, workplaces, academic literature, and the media. However, this position frequently reflects ideological views and lacks sufficient evidence. Statements that programming is hard can have obvious direct consequences. However, they can also convey more indirect messages—in effect sending signals that can have unintended consequences on students, educators, the community, and the discipline of computing itself. These are rarely considered.

Is programming hard or not? Current evidence is not compelling nor diverse enough to answer this question in general. More defensible (and likely honest) answers are: "it depends," and "both." Why then, is it so common to say that it is hard? Is it often said anecdotally because there is not that much evidence to support it? Because the evidence that does exist is difficult to understand? Could it be that it is just too convenient for motivating and justifying work? Is it that many want programming to seem hard, consciously, or unconsciously? Do tech companies and hiring managers depend on the image of programming being tough and elite? Is it too convenient for explaining phenomena whose true explanations remain elusive? Is it just an easy excuse for failure? Perpetuating this belief only serves to reinforce a shaky base of evidence that undermines any more rigorous evidence-based research. If we are going to make claims on the difficulty of programming, the community has a duty to provide robust empirical evidence from diverse contexts and state the findings responsibly.

Many current events and sociopolitical realities have caused us to question our educational practices recently. Considering the present global context, blanket messages that "programming is hard" seem outdated, unproductive, and likely unhelpful at best. At worst they could be truly harmful. We need to stop blaming programming for being hard and focus on making programming more accessible and enjoyable, for everyone.

References

1. Becker, B.A. et al. Compiler error messages considered unhelpful: The landscape of text-based programming error message research. In *Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education* (Aberdeen, Scotland UK) (ITiCSE-WGR '19). ACM, New York, NY, 2019, 177–210; https://bit.ly/2T97WUT
2. Birnbaum, L., Hambrusch, S. and Lewis, C. *Report on the CUE.NEXT Workshops.* Technical Report (2020); https://bit.ly/3x8ev8Q
3. Guzdial, M. Learner-centered design of computing education: Research on computing for everyone. *Synthesis Lectures on Human-Centered Informatics 8,* 6 (2015), 1–165.
4. Luxton-Reilly, A. Learning to program is easy. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education* (Arequipa, Peru) (ITiCSE '16). ACM, New York, NY, 2016, 284–289; https://bit.ly/3ivrKfM
5. Luxton-Reilly, A. et al. Introductory programming: A systematic literature review. In *Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education* (Larnaca, Cyprus) (ITiCSE 2018 Companion). ACM, New York, NY, 2018, 55–106; https://bit.ly/3v1D9qh
6. Nguyen, A. and Lewis, C.M. Competitive enrollment policies in computing departments negatively predict first-year students' sense of belonging, self-efficacy, and perception of department. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education* (Portland, OR, USA) (SIGCSE '20). ACM, New York, NY, 2020, 685–691; https://bit.ly/2TTr3Tl
7. Sime, M.E., Arblaster, A.T., and Green, T.R.G. Structuring the programmer's task. *Journal of Occupational Psychology 50,* 3 (1977), 205–216; https://bit.ly/3w4NNxL
8. Tedre, M. From a black art to a school subject: Computing education's search for status. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education* (Trondheim, Norway) (ITiCSE '20). ACM, New York, NY, 2020, 3–4; https://bit.ly/3v436po

Brett A. Becker (brett.becker@ucd.ie) is an assistant professor in the School of Computer Science, University College Dublin, Dublin, Ireland.