The *Communications* Web site, <http://cacm.acm.org>, features more than a dozen bloggers in the BLOG@CACM community. In each issue of *Communications*, we'll publish selected posts or excerpts.

Follow us on Twitter at <http://twitter.com/blogCACM>

DOI:10.1145/3358698

<http://cacm.acm.org/blogs/blog-cacm>

Pinning Down Variables, and Taking an Agile Approach

Robin K. Hill tries to better define variables, while Edwin Torres acknowledges he is his own greatest obstacle.

Robin K. Hill Variable Vagaries

May 19, 2019
<http://bit.ly/2YHWMYc>

A few months ago, I wrote in this space some speculation under the question, “What is a Variable?”³ and would now like to explore that question a bit further. The *Dictionary of Computer Science*¹ says that a variable is:

1. A unit of storage that can be modified during program execution, usually by assignment or read operations. A variable is generally denoted by an identifier or by a name.
2. The name that denotes a modifiable unit of storage.
3. See *parameter*.
4. (in logic) a name that can stand for any of a possibly infinite set of values.

For someone who appreciates a pesky problem, this is great stuff. Alternative 3 passes the buck, alternative 4 doesn't get us very far, and alternatives 1 and 2 are the two of the views we are trying to pin down and reconcile. (The reader should not infer con-

demnation of this definition or promise of a better one.)

That piece on variables in January garnered some comments. Vincent Di Carlo mentioned the work of Simon Funk, with references, who observes and illustrates some of the difficulties connected to regarding a variable as a container, and the use of predication to alleviate that difficulty.²

Peter O'Hearn mentioned Christopher Strachey, for which I commended and thanked him. But on reflection—that wasn't enough. Strachey's remarks on this subject deserve more attention.⁵ He brings up the complexity of the L-value (the address, or location; the variable identifier, in other words, on the left side of an assignment). He shows expressions in that place, and explicitly allows functions. We are completely used to the left-hand side of an assignment statement sporting an expression like $A[i]$, an array name with an index to a particular element. This, of course, requires evaluation; in fact, this L-value is a function. Examples of L-values such as $A[i+1]$ make that even

more cogent. The variable itself varies, in other words, an affordance that has been played down in our more mechanical and precise development of programming languages.

This might suggest something about the informal view I attempt to exhibit. Here is a lightly fictionalized story of variables from my small town, where everybody knows the local business-people.

I said to a friend, “I made a mistake—the sewing machine repairman's name is not Victor, but I can't remember what it is.” She said (now read carefully), “No, it's not not Victor. It's actually not Mercer.” Right, of course ... Victor was the repairman for vacuum cleaners, in the same shop, before then. But Mercer was the proprietor of the sewing shop, not the repairman. And she knew I was thinking of Mercer. As for the actual sewing machine repairman who was working for Mercer—let's call him X—he plays no part, and I have no clue as to his real name, except that it was neither Victor nor Mercer.

A gross translation of the pertinent sentences looks like this:

$\text{Not}(X) = \text{Victor}$ We're calling this false!
 $\text{Not}(X) = \text{Mercer}$ We're calling this true!

I refrain from expressing those in any notation approaching symbolic logic for fear of inducing some sort of modeling monstrosity. What is the variable at play here? It's X only in the sense that it's not X is the foil. If X were the principal, we could reason this way:

$X \neq \text{Victor}$
 $X \neq \text{Mercer}$

Both are true, so that the imposition of standard semantics is not particularly revealing, and does not explain in what sense my friend was right. That's not what I was saying in the story; rather, I was shamelessly assigning to the L-value $\text{Not}(X)$. As I recall, I had grasped the presence of negation when I entered the shop, once or twice, and called the repairman "Mr. Mercer," to which he finally replied, "I'm not Mercer. He owns this place." I had forgotten the details, but recalled the negation, and misdirected it. Does recalling it mean the negation itself is the value of some variable, or is the negation a variable with a name as its value?

I offer no answer to that. It would be interesting as an exercise in formal representation under the laws of some system, perhaps generating a shadow referent for the placeholder that eventually resolves correctly into Mercer on its way to expressing a lack of identifier for X . The predications discussed by Funk might show the way. Yet the remarkable thing is that we tolerate, communicate, exchange, and exploit all those vagaries. We deal with them with no apparent discomfort, but rather recognition of the quirkiness with a laugh. William Kent makes the point that, even though we cannot draw a principled distinction between the attribute and the relationship, the terms are still useful, which might be the most intriguing result of all.⁴

So when our students ask what is a variable, we could state one of the variants given by the *Dictionary of Computer Science*, or something simple like "an

expression for a memory location." They hardly ever ask. They learn it by exposure and experience, like we did.

Let's leave the last word to Strachey, in support of the motto he suggests:

If we attempt to formalize our ideas before we have really sorted out the important concepts, the result, though possibly rigorous, is of very little value—indeed, it may well do more harm than good by making it harder to discover the really important concepts. Our motto should be 'No axiomatization without insight'.

References

1. Butterfield, A., and Ngondi, G.E., Editors. *A Dictionary of Computer Science*. Oxford University Press, 7th edition, 2016.
2. Funk, S., *On Knowledge Representation*, 2013
3. Hill, R.K., January 31, 2019, "What is a Variable?" BLOG@CACM, <http://bit.ly/2KdGzCa>
4. Kent, W. *Data and Reality*. North-Holland Publishing Company. Chapter 5.0. Attributes, 1978.
5. Strachey, C., *Fundamental concepts in programming languages*. These lecture notes for the International Summer School in Computer Programming, 1967, are published in *Higher-Order and Symbolic Computation*, 13, 11–49, 2000, Kluwer Academic Publishers.

Edwin Torres
An Agile Approach to Learning Programming
<http://bit.ly/2NqVQL4>
 March 29, 2019

I recently read a book called *Rework* (<https://amzn.to/2zgVEgc>) and had a revelation. I have been an inefficient software engineer and flat-out showstopper at times throughout my career. Like many previous books, *Rework* claims you *can* create your own business. What makes this approach different is that it shows you that most of the obstacles are *you*. To oversimplify, just make progress, inch forward, build something that works and refine later. In other words, take an Agile approach. *Rework* convinced me my obstacles are not really obstacles. Now my new business venture is almost a reality. Finally.

I quickly realized the lessons I learned from *Rework* are applicable to other areas of my life. One such area is teaching. I teach programming to students and industry professionals. I often see new programmers struggle with basic, fundamental concepts. I have found that taking an Agile approach to teaching these students is extremely effective.

First, most programming books, even beginner books, are overwhelming. New programmers find it intimi-

dating when faced with everything a programming language has to offer. In reality, a programmer only needs a subset of commands to complete a given task. And it is not necessary to learn multiple ways to complete a task, when all you need is one. So I wrote *The Super Simple Programming Book* (<https://amzn.to/2zeElMv>) to teach basic programming concepts using the Python programming language. The book is a quick read, with lots of short, clear programming examples. Its purpose is to give a brief introduction, teach fundamental programming concepts, and help the programmer quickly move on to more advanced concepts. This is iteration, quick learning, and progress, just like Agile and *Rework*.

Second, my classes incorporate elements of the Agile methodology. A class is like a sprint planning meeting. I describe the content and assignments for the week, and there is a classroom discussion. At the end of the class, students begin the sprint (that is, learning). We even have a Slack (<https://slack.com/>) team to conduct real-time persistent chats outside of the classroom. Students may submit assignments (such as prototypes) early, get feedback, and resubmit for full credit. At the start of the next class, there is a sprint retrospective; students describe what went well and what was difficult. This is also an opportunity for me to enhance course materials for the next class. And the process repeats (iterative).

My teaching method loosely follows the Agile method, but it is Agile, nonetheless. Ever since I started using this approach, I have noticed a significant improvement in grades and student attitudes. Students often say things like, "your class makes programming easy" or "I never understood programming until now." Agile lets you learn as you go and build on that learning. It has worked well in my classes.

Robin K. Hill is a lecturer in the Department of Computer Science and an affiliate of both the Department of Philosophy and Religious Studies and the Wyoming Institute for Humanities Research at the University of Wyoming. **Edwin Torres** is a full-time software engineer at The MITRE Corporation and an adjunct professor of computer science at Monmouth University.