Jean E. Sammet - Programming Languages: History and Future

Comments on:

Sammet, Jean E. (1972). Programming Language: History and Future. *Communications of the ACM* 15(7), July 1972. pp. 601-610.

The object oriented languages Simula and SmallTalk were just being developed while this paper was written. In [4] the author mentions that developments of programming languages will be guided by the aim to make the communication between a computer and a user easier. In the next few decades object oriented paradigm was popularly accepted as an important programming paradigm. Has this paradigm made such communication any easier?

The notion of “user” has clearly changed significantly since 1970. I’m not sure that the comment makes as much sense. I’m also not sure that OOP, at least as it is practiced today, outside of the Smalltalk/Squeak community, really meets those standards.

Sammet in the article on page 603 says that importance and wide use are not the same, how and why is that the case for programming languages?

I think she makes the answer pretty clear - she defines importance in terms of the effect the language had on the computer science community, primarily the languages community. Wide use is just what it sounds like - How many programmers use it. In more modern realms, we might argue that Windows broke no new ground in operating system or user interface design (okay, at least for the first few versions, it’s clear that it went backwards). Nonetheless, it is still the most widely-used OS.

It was pretty interesting to read an article predicting the future, which is in fact the present.

I’m not sure what you mean by “in fact the present”. Are you claiming that Sammet’s predictions were correct, or simply indicating that Sammet’s future is our present? A clearer exposition of these issues would strengthen this comment.

While the day of asking the computer to “COMPUTE THE PAYROLL FOR ‘MY COMPANY’ is at least one or two decades in the future ...” My brother just asked if I could write a program to do that for his company the other day. I guess the author was wrong on that one.

Well, depending on how the data is entered, some SQL systems can do something similar.

How does the definition of programming languages/higher level languages compare to our own? The author has in there that there is good potential for conversion to other computers. This implies to me that he isn’t including the kind of languages that are used to represent algorithms as we discussed in class. What about the languages which are modeling problems for other people in addition to those used for computers? I also don’t quite understand what he means by part 4 of the definition, "there is a notation which is closer to the original problem than assembly language would be."

We will be discussing most of these issues in class. By "conversion to other computers", she *means that we can compile the program for multiple architectures.*

As a matter of curiosity, what ever did happen to FORTRAN and COBOL. The author goes on about how they'll keep being used in the future but now they're totally shunned, insofar as I can tell. What about them made them be so outmoded?

Sammet claimed that they'd last five or ten more years. Both are still going fairly strong. There's a huge amount of legacy COBOL still being used. Right before Y2K, COBOL programmers were in great demand for just that reason. Lots of Physicists still program in something called Fortran, although we might argue that it's a substantially different language.

It is difficult to write a history of programming language because of the number of stages a language goes through. The author spends a little more time on the 1950s and 60s than wikipedia and stresses its importance. Looking to the future from 1972, the author hoped that languages would develop to the point that they can understand English or at least user defined languages. He wanted less structure in the language and more control for the computer in how it performs tasks.

While I appreciate the summary, I want you to reflect, not just summarize.

So far, all the Programming Languages are alphabetic. Has any one thought about using representive languages? symbolic? like most ancient coutries' languages?

APL was symbolic (using Greek characters). It ended up being a bit too terse for ost people to read it. You'll see somewhere that it is considered a bit of a "write-only" language. One might argue that the code part of most regular expressions is not alphabetic, but rather symbolic.

It seems like all three articles skirt the question of what qualifies as a "programming language". Is this term defined clearly? And why do assembly languages not seem to count as programming languages?

Well, Sammet tries a definition with four criteria. In some sense, I think it's a turf issue. Assembly language (and machine language) are really architecture issues. High-level languages attempt more to separate themselves from the underlying machine.

The idea of writing a computer language is overwhelming to me, and so I am astounded by the number of languages Sammet claims had been created by 1972. It's also incredible that some languages developed then are still in use.

However, I don't think Sammet's "Reasons for Importance of a Language" are extensible to today. For example, there are languages which don't satisfy her "technically new" criteria but are still in heavy use on legacy systems because it is easier than upgrading.

I am curious if a paper written in 1972, before Internet culture, is still entirely applicable.

My intent is not for you to take everything you read in these papers as absolute truth. In fact, for this paper, I wanted yo to reflect on what has and has not changed.

I found her "snob appeal" comment interesting-- I wonder what she would think of the current computing culture, with language fanboyism and implicit or explicit programmer hierarchies `http://www.hermann-uwe.de/files/images/programmer_hierarchy.png`.

I think she already mentions fanatics. You could read some of her recent writings on languages; she's still active in the history of programming languages community (or at least was a few years ago).

It seems that a lot of effort was spent creating languages that had different features rather then being truly innovative. One language would make one set of design decision relating to elements like garbage collection, use of "Go To" statements, strong typing, or the use of pointers, while another language would make a different set of design decision, without either language providing any really new or useful improvements. It seems from the reading that lots of people agree on where they want programming languages to go, but there isn't much agreement on how to get there.

Well, Sammet does argue that few languages provided truly innovative things. (See her list of languages that fit criterion 1.) I would also suggest that different designers have very different views on where languages should go. We'll return to that issue in the coming weeks.

I liked the third reading because it cleared up the key concepts related to computer languages. It also ties in proving the correctness of programs (discussed in CSC 341). Even though it is pretty outdated, it lays a very good foundation for the study of programming languagues.

In what way does it lay a good foundation?

It is quite interesting that the first programming languages focused on list processing and business related tasks rather than scientific computations. I would have figured that the scientific community would have widely adopted computers before businesses did and therefore a consistent way to express scientific formulae across systems would have been developed first. The text does mention that FORTRAN was quickly adopted by the scientific community, but I don't see why that would be the case given that the paper does not mention it as a language which is good at expressing scientific computations.

Well, there does seem to be some bias to Sammet's description. Just so you know, FORTRAN really was designed for scientific computation, so it makes sense that it was adapted. (I'd also argue that Algol was general purpose.) The Plankalkul (sp?) was also clearly numeric.

Art vs. Science

You being an experienced programmer, how much do you think is programming an art and how much science?

I think algorithm design is an art. You can't just sit down and follow a set of rules to get a product. I think that really good software architecture is an art, just as real architecture can be. However, much programming is like much engineering - if you follow the rules and guidelines, it should be straightforward, and that's what people tend to mean when they say that it's a science.

“Vitually everyone agrees that today programming is an art, not a science.” I think if I asked anyone today, they would say programming wasn’t an art, but maybe a science. The author thought programming could become a science, but I I don’t believe that people actually thought programming was an art. Was that really true or did he mean art in the sense of creation without following strict scientific principles like the scientific method?

There is a small, but vocal, group in the CS community who consider good programming an art. There’s even someone who is trying to put together a Master’s of Fine Arts in Computer Programming. We’ll read a bit by that community. I expect that she *really did mean the claim at the time; in the early 1970’s, there were not a clear set of principles by which you designed programs, making it much more of an art than a science. Also, fitting an algorithm into a restricted space was clearly an art. If you question the view, simply think about the title of Knuth’s great work, The* ***Art*** *of Computer Programming.*