

Wikipedia - History of Programming Languages

Comments on:

Wikipedia (2007). History of Programming Languages. Web resource, available at http://en.wikipedia.org/w/index.php?title=History_of_programming_languages. (Dated 1 January 2007; visited 19 January 2007.)

It seems like the term “programming language” is loosely defined. This has created a problem for historians to clearly outline the history of programming languages. If programming language is a grammar for instructing machines (like we discussed in class), then (the author claims that) even the Difference Engine and Jacquard loom had some kind of programming language to instruct respective machines. In that case is there a more specific definition of the kind of programming languages that we will be studying about? Or should the historians record the development of programming languages prior to 1940s? Is it appropriate to say that programming language is a grammar for instructing “electronically powered” machines?

Yeah, that seems like a good, open-ended question to discuss in class.

Why was it in the case of the 1980s that rather than inventing new paradigms, all of the movements elaborated upon the ideas invented in the previous decade?

It may be that all of the primary paradigms were discovered. Much of the language research in the 1980's emphasized exploring those paradigms further, formalizing them, and formalizing other, related issues, such as types.

It seems that a lot of effort was spent creating languages that had different features rather than 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 decisions, 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.

Boy, 1967-1978 are described as a time of significant differences. You need to read it a bit more carefully.

I didn't know 50 percent of what was in this article. It was pretty interesting.

Your lack of knowledge does not a substantive comment make.

Was the guy who invented Plankalkul really named Konrad Zuse? That name is too awesome.

Subtleties of inventor's names is also not very substantive. And yes, that was his name.

The article described the design of Java as being “a more conservative version of ideas explored many years earlier in the Smalltalk community”. What makes the design in Java more conservative than Smalltalk?

Well, Smalltalk was fairly liberal. In part, Alan Kay (it's primary architect) had a much broader vision of computing than we see in Java. For example, Kay wanted Smalltalk to be accessible to children, and to encourage exploration. Java is clearly targetted only towards professional programmers. Kay wanted Smalltalk to be indistinguishable from the computer; Java clearly builds programs that run on computers. Smalltalk comes from a design philosophy that says “trust the programmer”. Java clearly comes from a philosophy that says “restrict programmers, or they will make mistakes”.

We will return to these differences as we start to explore the philosophy of Lisp-like languages.

Do you know why Konrad Zuse was not able to implement his programming lanugage? What inspired him to develop a programming language?

Implementing a programming language is hard. The first real high-level lang uages (I think of Fortran) required huge teams to develop, including some of the brightest minds of the time. Zuse was clearly bright, but I do not think he ha d a sufficiently large team. He also had his computer destroyed and did not seem to have much external support.

I expect that his inspiration was the obvious one - he wanted to better describe computation for the Z3.

If you really are interested, Zue and his language would be an excellent topic for a presentation.

Wikipedia's article was slightly more detailed than BYTE's history. It outlines several stages in the development of programming languages. The main types of languages were developed in the 1970s that were mostly just expanded upon later. The internet launched Java and was the big change of the 1990s.

No response seems necessary.

What is BNF and how important was it in developing languages?

Backus-Naur Form. A technique for describing the syntax of programming languages, now amenable to automated parser generation. Almost every modern language gets a BNF syntax description.

About the current trends, what are extended static checking, information flow control, static thread safty, mixins, delegates, component-oriented.

Ah, so many opportunities for you to use Google and Wikipedia. Also so many options for presentations at the end of the semester.

Why Open Source? What's the benefit?

Such a politically loaded question. One clear benefit is that you have other eyes to comment and work on the language but you can often have one or two minds directly the project. I'm sure that Mr. Stone will tell you others.

The articles all mention when languages were made to implement a certain paradigm (e.g. Smalltalk in the mid 1970s to implement OO). How long have the paradigms been around? What were the obstacles that prevented people from developing OO oriented languages before that?

The paradigm and the language typically went hand-in-hand. A paradigm was developed, and the language provided a proof of concept. (Alternately, upon reflecting on the design of the new language, computer scientists identified general principles.) OOP was developed in the mid-1960's with Simula, but lots of the great ideas, and the generalization of those ideas, does come from Smalltalk.

But the machines were also a limiting factor. Remember, 64K of RAM was a huge amount, even in the early 1980's. Think about fitting a whole WIMP interface in such a space (which Smalltalk did).

This article states that the 1980's saw a reduction in the number of popular programming languages and instead focused on improving the performance of both compilers and compiled languages. What were the factors that sparked this change?

Well, the authors do suggest one reason - new architectures suggested that there should be more emphasis on compilers. I expect that as computers became more widespread, the need for programmers to transfer expertise to other domains made many industries choose languages that they expected programmers would know. (Some argue that's why we see so much Java these days - not because it's necessarily the right language for the task, but because every company knows that it can easily hire Java programmers.) Beyond that, it's outside my knowledge.