Paul Graham - Revenge of the Nerds

Comments on:

Graham, Paul (2002). Revenge of the Nerds. Online resource available at http: //www.paulgraham.com/icad.html.

Paul Graham talks about macros in LISP in his article. On trying to look for definition of macros I ran into the following quote at http://www.apl.jhu.edu/~hall/Lisp-Notes/Macros.html:

> Lisp functions take Lisp values as input and return Lisp values. They are executed at run-time. Lisp macros take Lisp code as input, and return Lisp code. They are executed at compiler pre-processor time, just like in C. The resultant code gets executed at run-time.

In other words, isn’t the concept of macros in LISP the core idea of the funtional programming paradigm although it generalizes the concept of higher order procedures? How is it different otherwise?

Not quite. Macros work with the source code. Higher-order procedures cannot actually get into the internals of a procedures.

This paper makes a great argument as to why scheme is so powerful. The constant Dilbert reference only drives his point. He also reiterates the point made in other readings that the language that is fashionable at a certain moment is not necessarily the most powerful one. Aside from shorter code, I don’t really understand what he means by power? You have mentioned in class that pretty much all modern languages have the same computational power. So what makes scheme more powerful that the fact that you will take less time typing your code? Does it perhaps excersise the brain in a different way that other languages?

By power, I think he means the number of things that you can express easily. While you can express many Lisp concepts in C or Java, it takes a lot more effort. The “less typing” comment is also correct - Lisp is powerful in that it takes significantly fewer “words” to say the same thing. And yes, it does seem to exercise your brain differently.

This article argues that Lisp is generally superior to languages like C++ or Java and that the primary reason the other languages remain in common use is because of historical momentum. In "The End of History and the Last Programming Language" Richard Gabriel argued that the "mathematical sophistication" required by languages such as Lisp was a weakness.

The issue seems to be, is Lisp a better language, or do better programmers just use Lisp? If Lisp is a superior language, and the primary reason it isn’t widely used is due to historical momentum being against it, wouldn’t schools that are capable of producing programmers who posses "mathematical sophistication" focus on teaching them how to use Lisp?

After all, if historical momentum is against Lisp, the academic world seems the most likely candidate for reversing the process. Unlike industry, the academic world doesn’t have the pointy haired boss who wants to protect themselves by blindly following the standard -- right?

Given how much trouble it seems most students have understanding recursion, I'd say that the "mathematical sophistication" thing still works against us. Gabriel also suggests another reason - most people who just do a little programming in Scheme or Lisp don't really "get it". So we get a cyclic process where very few faculty know Lisp because very few of their faculty know Lisp. The Lisp/Scheme crowd has also been a bit offensive at times, which drives many people away. Plus, there's the whole lack of strong typing thing.

Is it true that the only thing that differentiates one programming language from another from a programmer's viewpoint is the relative simplicity in solving problems using the language?

No, many programmers care about efficiency and portability, too.

If LISP was so great a language, was there any need for any more languages? One may argue that there was need for other programming paradigms such as OO hence such languages as C++ and Java. But LISP does have an OO variant, CLOS.

Different programmers think differently, so if we care about relatively simplicity, then the answer is yes. It's also the case that early Lisp systems couldn't be used for a number of applications because people worried about efficiency, particularly with the garbage collection system. And CLOS was a relatively late addition.

I find it difficult to be convinced that the main reason all programming languages are considered equivalent is because it is a comfortable idea for everyone.

It's not just a comfortable idea. Church proved that all sufficiently powerful models of computation are equivalent.

Further, later on the writer says that Lisp and Fortran were the trunks of two seperate evolutionary trees and that the two trees have been converging ever since. Thus, he is suggesting that the languages are similar and have been converging for years, without mentioning anywhere that they are similar just because it makes things simpler.

Um, there's a big difference between suggesting that they're converging and suggesting that they're similar. Even if they're very different, each language can adapt ideas from the other language, which then brings them closer together. For example, Fortran added recursion and Lisp added a real number data type.

Graham says, "Code size is important, because the time it takes to write a program depends mostly on its length. If your program would be three times as long in another language, it will take three times as long to write...." Do you think this is necessarily true? It seems like difficulty of implementing the code would mostly cause the code to take longer to write, not how many characters are needed to express the same ideas. Would Hoare disagree with Graham?

The studies I've seen show that programmers are remarkably consistent in the number of lines they produce each day, and that that number is independent of the language used. I'm not sure what Hoare would say.

How does a less powerful language use significantly less lines of code than a more powerful language? I'm assuming that his statement holds true even for more difficult and larger tasks. But aren't more powerful languages suppose to make the more difficult tasks easier?

Um, as far as I can tell, Graham claims that more powerful languages, not less powerful, use significantly fewer lines of code.

The argument appears to be made that you can never justify not using LISP. It may be okay to use another language so long as you're doing something simple, but if you ever need to get a bigger project done you should be using LISP because it's better. If that were true, why is LISP not used for writing more programs? The author offers up the two partial explanations. The first is that you want your code to be interoperable with whatever operating system you are using. The second is that people tend to be swayed by conventions. But then he counters those again, by showing the differential in time and code size for those who use LISP vs those who use lower level languages.

I'm not sure how "Lisp is more efficient" counters "I want to use what other people use" or "I want to use what I know". And, as far as I can tell, those are the key points. In addition, as Gabriel mentions, most people who just do a little bit of programming in Lisp not only fail to understand how to use it well, they use it incredibly badly (e.g., not realizing that `reverse` *is an O(n) operation).*

I don't understand the part in which the paper talks about some issues with eval. Could you explain the eval function a little bit as well as the related stories.

`eval` *takes an s-expression as a parameter and evaluates it. This means that you can make a structure that represents code and then immediately evaluate it. Here's a very simple example to show what's going on.*

```
> (define silly (list 'cons 1 'null))
> silly
(cons 1 null)
> (eval silly)
(1)
```

The power of this is, of course, that you can generate code on the fly (based on user input, but turning strings into symbols, ...).

The author, did a great job at explaining the power of LISP. But I also felt that he assumed most programers can easily write and understand very concise code. From a past reading, we learned that java was created with a goal of appealing to a wide audience, perhaps that could partially explian why it is not as concise as LISP. Perhaps some programers choose a weaker language because they find it easier to master and not solely because their bosses say so.

A very nice response to one of the questions above.

Having people who do not understand technology make decisions about it is a really bad idea. There should be a program related to administering technological businsses so that managers might become better decision makers in their field.

There are such programs. It doesn’t necessarily help.