# Programming Languages

Benedict Dugan © 1994

> The loss of material space leads to the government of nothing but time... The violence of speed has become both the location and the law, the world's destiny and its destination.
> -- Paul Virillio

I wish to look at programming languages from the perspective of technology as a social construct. In particular, I want to examine the social and economic forces which drive the acceptance or non-acceptance of certain programming languages. The argument that I will make is that the decision to accept a given programming language is not necessarily based on what is the "best" language for the job, but rather a wide variety of other factors. In conclusion, I will comment on what I see as the disturbing implications of the status quo.

## How do we choose languages?

I wish to answer this question: Given the huge variety of languages to choose from, why and how do we chose the ones we do? In examining this question, my starting point will be the recent decision made by our department to move from Ada to C/C++ as the language of choice for the introductory programming classes. The purpose of this paper is not to debate the relative pedagogical virtues of Ada and C. Given these two choices, I would say that there is in fact no debate and that Ada comes up as the hands down winner. So, unless I have really missed something, I ask: If it is universally understood that C is not a good teaching language, why have we chosen to use it as the foundation of our introductory sequence? To answer this question, I will examine some of the arguments which have been made in favor of the transition.

### *It will get the engineering school off of our back and relieve us of the job of teaching FORTRAN.*

Currently, one to two weeks of the first quarter programming course are wasted on teaching the student's FORTRAN, presumably to appease the powers that be in the Engineering School. Recall Dykstra's comment:

> The sooner we can forget that FORTRAN ever existed, the better, for as a vehicle of thought it is no longer adequate: it wastes our brainpower, and it is too risky and therefore expensive to use. FORTRAN's tragic fate has been its wide acceptance, mentally chaining thousands and thousands of programmers to our past mistakes.

Clearly, teaching FORTRAN is a mistake. The argument says that by teaching C, we can meet the demands voiced by the engineers for a "useful" language, thereby ridding us of the curse of FORTRAN. From this perspective, the decision to teach C is the result of a struggle of priorities between the Engineering School and the CS department. Notice that this particular argument makes no claims about the pedagogical virtues of using C.

*The students want to learn C.*

It is doubtful that this argument is really a factor. However, it is interesting to think about why the students all seem to want to learn C. They clearly are not interested in C because it has a reputation for reliability, or for being a particularly expressive environment. Again, there is no mention of pedagogical superiority; rather, I would argue that they are interested in C because it is a valuable skill which will be useful during their careers. It is considered an important skill because C is a dominant industry language.

*Because we use C in all of our higher level courses, it is to our benefit to teach it to them early on, rather than wasting time in a higher level course.*

This argument actually does address some pedagogical issues, namely that we can create a more consistent, far reaching teaching plan if we use just one language in all of our courses. This argument for a universal language has its dark side, however, in that it will lead to the creation of programmers and designers whose vision and capabilities are constrained by the fact that they have become comfortable with only one language, one paradigm of programming. So this argument is also dubious, at best.

The real question then becomes this: If the fact that C is used in higher level courses has determined the acceptance of C in introductory courses, we should ask why we chose to use C in the higher level courses to begin with. The answer to this question is many-faceted. First, C has a reputation for efficiency. Because C is speedy, it has become a language of choice for compiler writers, who want a speedy end product. In time-critical applications, C has become popular because it is decidedly easier to work with, more portable, and less error-prone than assembly language. Also, the symbiotic relationship between C and Unix, coupled with the early acceptance of the Unix OS throughout the universities, has assited C's ascension. Because most OS research (and by extension, Network research) takes place on Unix platforms, C has become the obvious language of choice in those domains as well. AI seems to have been the only area that has managed to resist the seemingly inevitable expansion of C into all aspects the computer science field. What I hope to have shown in this passage is that the decision to use a given language for a certain

application (teaching, in this example) is not necessarily based upon which is the "best" language for the job. It appears that the decision is often rooted in factors on the outside, such as industrial acceptance, reputation, or a simple acceptance of an apparent status quo. If we dig a little deeper, we will often find an emphasis on efficiency and speed (in as many ways as we can imagine: compile times, code generated, portability, etc.). The only real objection which is consistently voiced is that C is not a good language for software engineering, due to reliablity concerns (like those raised by Horning). However, C's most recent incarnation, C++, appears like it will come some distance towards silence some of these critics.

A Digression: What should programming languages be used for?

I would like to digress for a moment to present my view of programming languages. The the obvious answer is that programming languages are there to control the computer. This is certainly true, but there is another view. I hold that programming languages are far more than a tool for controlling computers. Instead, programming languages are ideologically laden technological artifacts. That they are technological artifacts is obvious, but the notion that they are ideologically laden may not be so clear to the reader. They are value laden because depending upon their structure, they can act as either the barrier between, or the conduit for, giving the public control and access of computer systems and all that entails. Knowing or not knowing the language quite literally translates (at some level) into having power or not. Programming languages - if they are "done right", according to Alan Kay - will become communication media which will amplify the human potential for creativity and understanding. The key thing, however, is that the languages are done "right." I do not wish to get into an argument about what programming languages should look like, but suffice it to say that I find the current offerings to be severely impoverished in terms of meeting the goals I have laid out above.

Why is this? There are at least two moderately successful languages which have come as close as any to being "done right." These are Smalltalk and Lisp. Even though neither of these languages are particularly popular, their best ideas most certainly are. The ideas of abstraction and object orientedness respectively, were arguable first instantiated in these two languages. While Smalltalk has not become a huge hit, and Lisp has languished in the AI labs of the world, their "good" qualities have become widely accepted and assimilated. Smalltalk brought about the revolution of "object orientation." What is interesting about this revolution, is that it is not at all what Kay et. al. had in mind when they went about designing that language. They hoped to create a language which was greater than itself, embodied a new design philosophy, was widely accessible, and most importantly, would foster (rather than hinder) the creative processes of the human mind. Instead, the notion of object orientation was viewed by the world at large as a great

way to reuse code and increase the efficiency of software development. Again, OO languages were viewed as "good" largely because of their potential for efficiency, and it is in this way that an OO extension to C, like C++ has the potential to silence many critics of that language. C++ retains most of the desirable (meaning fast) aspects of C, while adding the capabilities of object oriented programming to improve its efficiency as a software engineering tool.

So, Why does this Bother me?

I have painted a picture in which a quest for efficiency becomes the driving force behind decisions to accept programming languages. These decisions are made at the expense of good pedagogy at the very least, and the human potential for creativity at the very most. Social theorists such as Weber, Marcuse, Horkheimer, and Habermas and others have examined the history of rationalism and its coupling with capitalism and have painted a bleak picture of a dystopia in which the very fabric of everyday existence is subsumed under the drive for efficient production. In 1947, Max Horkheimer stated:

> Concepts have been reduced to summaries of the characteristics that several specimens have in common. By denoting similarity, concepts eliminate the bother of enumerating qualities and thus serve better to organize the material of knowledge. They are thought of as abbreviations of the items to which they refer. . . Concepts have become 'streamlined,' rationalized, labor-saving devices. . . thinking itself [has] been reduced to the level of industrial processes . . . in short, made part and parcel of production.

Horkheimer is referring of course to the application of the methods of the rationalistic tradition applied to the fabric of knowledge itself. As industrial society applies scientific rationalization to all regions of the social sphere, the structures of daily life become increasingly mechanized. The logical and frightening conclusion of this process, Horkheimer maintains, is the mechanization of knowledge and the very processes of thought themselves. The reason that I have quoted this passage is because of the way we can read it in the context of object oriented languages. It is interesting to note that object-oriented programming is most often promoted for its capacity for exactly this act of *streamlining* and *organizing* knowledge in a labor-saving manner, referred to by Horkheimer.

In the end, I don't think that we need to understand the obscure writings of various highfalutin German and French social theorists. I think that what is important is to understand what is driving us to accept and use certain programming languages. Obviously, we need to understand what features make a given language reliable, expressive, efficient, or dangerous. However, understanding these things means little if we don't understand the mechanisms of technology

transfer. Understanding the background forces that lead instituions and people to accept or not accept languages is vital because it gives us the ability to address these factors intelligently and bring about change in a positive direction. Currently, I see languages being accepted at the expense of good teaching practices, reliability, or good software engineering principles. At a higher, more philophical level, we should also understand that the particular languages we chose can greatly affect our potential for creativity. This said, I find it a great pity and a shame that our department (and many others around the nation) - whose stated purpose it is to foster education, exploration, and creativity - has so universally embraced a language which is inherently so counter to these goals.