BLOG@CACM

What Makes a Program Elegant?

By Robin K. Hill
October 11, 2016
Comments (2)

A subfield of philosophy is aesthetics, in which we attempt to understand beauty. Is beauty universal? Does it make us better people somehow? Why do we focus on beauty and not ugliness? A ready application of this question to computer science addresses program elegance. Most programmers, or so I believe, would agree that some programs are elegant, and that elegant programs are better than others, and experienced programmers, or so I believe, generally agree on which programs are elegant.

The criterion of efficiency looms large in production programming, and appears in comment on elegance on the Web, for instance by Perrin [Perrin 2006]. A program should be brief, but not a slave to brevity. An elegant design artifact is sleek and spare in its utility. An elegant program is minimally gratuitous. For examples in familiar problem areas, consider Binary Search (of an ordered sequence) as opposed to Sequential Search, or Quicksort as opposed to Insertion Sort [Gelernter 1998]. Sequential Search tediously examines each (ordered) item, but it doesn't have to; Bubble Sort tediously exchanges many items that will have to be moved again. To find the first *n* prime numbers, we can tediously test each for divisors or we can deploy the Sieve of Eratosthenes. Efficiency helps make the Sieve, Binary Search, and Quicksort elegant. We have our first criterion for elegance, **(1) minimality**, encompassing both shortness and simplicity.

Let us avoid the features of programs that depend entirely on source code syntax, or compilers, or I/O mechanisms, or memory handling. A program that minimizes temporary variables, directly evaluating expressions instead, is "better," but we do not address the question of aesthetics at that level, nor at the level of self-describing identifiers, nor documentation, nor modularity, nor design patterns. A program also becomes better as it includes more error-checking, but that does not strengthen, and may indeed weaken, its elegance, even as it enhances its quality.

Simplicity by itself can't be enough; Bubblesort is a simple program. (I would count Boyer-Moore String Search as elegant even though it's complicated.) Brevity by itself can't be enough; the C loop control `while(i++ < 10)` may be terse, excelling in brevity, but its elegance is debatable. I would call it, in the architectural sense, brutalism. Architecture provides nice analogues because it also strives to construct artifacts that meet specifications under material constraints, prizing especially those artifacts that manifest beauty as well [De Botton 2006].

A factor that looms larger in computer science than in architecture or other disciplines is correctness. A building may be regarded as elegant even if marginal parts of it are uncomfortable, but no program that does not work is regarded as elegant. This gives us another criterion, **(2) accomplishment**—the program does what it is supposed to do. Though included in the list of desiderata here, failure on that criterion is fatal, rather than detrimental.

The constraints under which programming is done impose a context without which the elegance cannot be appreciated. We must understand the problem, and the tools and materials, in order to appreciate the solution. Expertise is necessary. Examining many student programs over many years refines an appreciation ever more impressed by work that does it all with graceful assurance and economy. Elegance, therefore, is doubly relative—to the context of the work and to the background of the observer. Consider Bitmap Sort, as presented by Jon Bentley [Bentley 1983] in a classic "Programming Pearls" column, still worth study. To sort

n unique integers in a fixed range 0 to *m*, we can rearrange them through a comparison-based sort, such as Quicksort, or we can initialize a bit array, indexed by 0 to *m*, to *false*, and then, for each integer input, flip its bit to *true*. One pass through the resulting array, during which the indices of the *true* bits are output, gives us the sorted list. This is nice. This is elegant, even relative to Quicksort, but note that it only works on a set of unique values (as described); the recognition of situations that meet that restriction distinguishes the programmer of elegance.

We are ducking myriad hard questions about the implementations at various levels of translation, and whether they should count toward or against elegance. And we will continue to do so. In fact, what I have been describing is not programs in source code terms, but algorithms. Brevity, or minimality, is a salient feature of code, but a subtle feature of algorithms; what we want, of course, is minimality in terms of the *solution*, however that solution is expressed. Yet another more general concept of spareness is at play in elegance, something like restraint. This gives us a criterion of **(3) modesty**. An example that flouts it comes right off the very first page of another classic, Kernighan and Plauger's "Elements of Programming Style."

```
DO 14 I=1,N     DO 14 J=1,N 14 V(I,J) = (I/J)*(J/I)
```

This exploits the FORTRAN compiler's truncation of integer division results to populate a matrix *V* with zeroes everywhere except the diagonal, where the values are one; that is, it initializes *V* to the *NxN* identity matrix. This is clever, very clever, and very short. But... oh, dear... it's implementation-dependent, therefore fragile; it's obscure and ostentatious. Such virtuosity is unfortunate, yet so hard to resist. (Kernighan and Plauger propose the obvious initialization to zero throughout, followed by a loop that assigns the value one to each *V(N,N)*.)

What else counts? An elegant program confers a sense of satisfaction, of enlightenment. Let's call this criterion, especially characteristic of program artifacts, **(4) revelation**—the program shows us something new about its task, or brings to the fore something that we forgot. Eratosthenes's Sieve shows us, or reminds us, that multiples are the "not-primes." Bitmap Sort shows us, or reminds us, that the integers are already ordered; they come as a sequence, so the sorting can be accomplished by an indication of presence only. Boyer-Moore String Search shows us, or reminds us, that strings are just as distinct backward as they are forward.

The criteria for program elegance suggested here are **(1) minimality**, **(2) accomplishment**, **(3) modesty**, and **(4) revelation**, all rooted in the particulars of the problem. Are these criteria necessary? Sufficient? Inadequate? Because of heavy dependence on the problem at hand, sometimes with complex circumstances, a wide range of examples of elegant programs is hard to come by. What are the exemplars that stand out in your world?

References

Bentley, Jon, "Programming Pearls: Cracking the Oyster," 1983, Communications of the ACM 26:8.

De Botton, Alain, "The Architecture of Happiness," 2006, Pantheon Books.

Gelernter, David, "Machine Beauty: Elegance and the Heart of Technology," 1998, Perseus Books.

Kernighan, Brian, and Plauger, P.J., The Elements of Programming Style, 1978, McGraw-Hill-Book Company.

Perrin, Chad, "ITLOG Import: Elegance," 2006.

Robin K. Hill is *adjunct professor in the Department of Philosophy, and in the Wyoming Institute for Humanities Research, of the University of Wyoming. She has been a member of ACM since 1978.*

Comments

CACM Administrator

February 22, 2017 01:55

The following comment by Mario Beland regarding "What Makes A Program Elegant?" was submitted on February 22, 2017:

I have enjoyed the article of Ms Robin K. Hill, on beauty of programs. I recall the book of Edsger W. Dijkstra "Beauty is our Business". Being a programmer, I have been using regularly his program "Updating a sequential file" which appeared in his book "A Discipline of Programming, Prentice-Hall, 1976. It is a beautiful program indeed!

--Mario Beland

Robin Hill

February 28, 2017 04:34

Thank you for your thoughts, Mr. Beland, and for inspiring me to revisit Dijkstra. I recall my chagrin years ago when I was asked to teach a course on File Structures, which I considered just too, too dull and mundane. Yet after that, I had a new appreciation for the elegance of the simple algorithms for merging and indexing.

Displaying **all 2** comments