Worse is Better is Worse

Nickieben Bourbaki[1]

In June 1991, AI Expert published a paper by my friend Richard Gabriel entitled "Lisp: Good News, Bad News, How to Win Big." One of the nice things about the article—really the only nice thing—was that it was an attention-grabber. I hate to say it, but Richard was dangerously wrong to publish that paper. Here is the story.

In the Spring of 1989, he and I and a few of his friends were chatting about why Lisp technology had not caught on in the mainstream and why C and Unix did. Richard, being who he is, remarked that the reason was that "worse is better." Clearly from the way he said it, it was a slogan he had just made up. Again, being who he is, he went on to try to justify it. I've always told him that his penchant for trying to argue any side of a point would get him in trouble.

A few months later, Europal—the European Conference on the Practical Applications of Lisp—asked him to be the keynote speaker for their March 1990 conference, and he accepted. Since he didn't have anything better to write about, he tried to see if he could turn his fanciful argument about worse-is-better into a legitimate paper. He produced the paper and presented it to the conference. As he finished his speech there was a stunned silence followed by strong disagreement from such people as Gerry Sussman. The local press took his speech as an affirmation of the widely held belief that Richard thought Lisp was dead. Several headlines proclaimed it.

So Richard buried the paper in his directory at work until one day about a year later—hackers being hackers—a young, new employee found it and sent the worse-is-better section of it to his friend at CMU, who sent it to two friends of his at Bell Labs, and so on. Pretty soon people all over the net were reading it and Richard was getting 10 requests a day for the full paper. Larges companies (with 3-letter names) were using the paper as a talking piece for discussions on how to design software for the '90's.

Richard got the urge to publish it more widely, and AI Expert went along.

This is a bad paper, and since no one else has stepped forward to refute it, I feel I must, despite Richard being a long-time friend—the friend who talked AI Expert into letting me write these columns.

There are two parts to my criticism. The first concerns the argument about worse is better itself, and the second concerns the effect of his argument and conclusions on people who are just entering a career in software design or who are students.

I'll start by recapping his argument. There are two constrasting software design philosophies, called the right thing and worse is better. With the right thing, designers are equally concerned with simplicity, correctness, consistency, and completeness. With worse is better, designers are almost exclu-

---

1.Nick wrote this one afternoon while watching the snow spiral down into the swirling Merrimack River, past the deadly inviting Rocks Village Bridge—while looking out his back window.

sively concerned with implementation simplicity, and will work on correctness, consistency, and completeness only enough to get the job done, sacrificing any of these other qualities for simplicity.

He gives an example of the two philosophies: how to handle user interrupts in system calls. Normally when there is an interrupt, the program counter (PC) at which to resume is stored somewhere so when the interrupt is dismissed, execution can resume, and in this case the PC plus a few state bits can accurately capture the state of the program so that the interrupt can excecute essentially arbitrary code. When the program is in a system call, however, the PC and the few state bits are not enough, because, for example, temporary resources might have be allocated by the system, and the state of these resources must be saved as part of the program state. The right-thing solution is to either press forward or back out of the call. The worse-is-better solution is to always return a flag that tells whether the system actually did the system call. A correct program, then, would test this flag.

The right-thing code for the solution takes a hundred or so pages of assembly language to accomplish, and usually the code is littered with places where some small mistake in the coding can cause serious errors. The worse-is-better code is hardly anything at all, and the user is required to be part of the solution. Have you ever hit control-C while a Unix program is running and notice that if you resume it, the program crashes? This is because the program does not test that some system call failed.

Now, offhand the worse-is-better solution seems stupid, because this means every system call should look like this:

```
try-again system-call
          when-failed
           go try-again
```

More on how stupid this really is later.

The argument, though, is that programs designed and implemented using worse is better will be written more quickly than right-thing versions, will run on a wider range of computers, will be easily portable, will be accepted more quickly if good enough, will be eventually improved, and will, generally, demonstrate better survival characteristics than right-thing programs. Worse-is-better programs are like viruses that spread quickly and are soon pervasive.

Of course, the two examples are Unix and C. Richard goes on to call the worse is better approach the "New Jersey" style, and I have heard him on many occasions say "what do you expect from an operating system designed and implemented in New Jersey!"

Before I launch into my rebuttal, let me point out the context of the argument. It is presented in an article about Lisp and its ups and downs. The real point of the article is to try to see where Lisp went wrong, as if that were a valid exploration. The worse-is-better argument seems to be a sour grapes look at the perceived villain in the fortunes of Lisp. There is no such villain, and the worse-is-better argument is just so much vitriol poured onto the page.

My first rebuttal is that there really isn't a worse-is-better design philosophy. The caricature of what this philosophy is so ridiculous that no one would ever express, espouse, or follow it. There are programs that appear to have been designed on the fly or whose designs have been subject to certain hard limits—like the size of the target machine or resources available to do the coding—but no one

has ever used this philosophy per se. The dichotomy that might exist is that of right-thing design versus no design. This is very different.

Second, can anyone really ever achieve the goals of the right thing? Right thing is pie-in-sky and is to be put in the same drawer with apple pie and mom. There are always tradeoffs in design and implementation, and no project (except Algol 68?) ever really tried to be pure right thing. Therefore there cannot be pure examples, only examples of varying degrees of tradeoff, almost always dictated by outside constraints, such as machine size.

Third, the best understanding of what is meant by worse is better and the right thing is presented in the PC-losering example that I described above. What is so right about the right-thing solution and so worse about the worse-is-better solution? The right-thing solution is lengthy and therefore likely to have mistakes in its implementation. Computer hardware is generally not designed to manipulate interrupts in multi-level programs—programs like a user program calling system routines—so there is a mismatch that must be bridged with software. If there is a bug in the PC-losering code, is this a better solution than handing the problem off to the user? The complexity of the right-thing solution is so great that many consider it nearly impossible to get it right.

And, if the user program is simple and can be preceeded with the advice to not hit control-C, what's wrong with sprinkling it with unprotected system calls? And what is so bad about writing protected system calls? Perhaps the interface to a worse-is-better system is not uniform because some system calls require protection and others don't, but certainly this is a small inconsistency whose correction requires heroic efforts. Perhaps a good tradeoff is what it is.

Buried under implication in Richard's paper is the thought that there is or was a choice between a right-thing and a worse-is-better solution in some area of computing important to a large group. The next few points argue that there has never been such a competition, and that within the context of the worse-is-better systems, these system were the right thing.

Fourth, how bad is Unix? We need to look at it in its historical context. Unix was written for the PDP-11. The primary alternative operating system on the PDP-11 was RT-11, which was vastly worse than Unix. For this machine, Unix was the right thing solution. The other, better operating systems that Richard is talking about were on the PDP-10 which did not compete with the PDP-11. Later the PDP-11 was replaced by the Vax, which also ran Unix. The people who used PDP-10's who needed to move to Vaxes had no other choice in machine or operating system. Therefore, there was never a head-to-head competition between a right-thing solution and a worse-is-better one.

Fifth, C is coupled with Unix in the worse-is-better scenario, and can anyone seriously propose Lisp as the right-thing alternative? Lisp, face it, is used for advanced research and development in AI and other esoteric areas. It has weird syntax, and almost all other computer languages share a non-Lispy syntax. Syntax, folks, is religion, and Lisp is the wrong one. Lisp is used by weirdos who do weirdo science.

Sixth, the role of the host computer as determiner of success has been conveniently ignored in the worse-is-better argument.

The successes of the operating systems on the PDP-10 versus the PDP-11 were more directly influenced by the success of their host machines. PDP-11's were cheap and easy to buy with small

research grants or with small development budgets. Vaxes were more expensive, but still 5 times cheaper than the PDP-10, which had an address-space limitation never really fixed (though the PDP-10 had 36-bit words, the address space was limited to 18 bits).

The PDP-10, which hosted a right-thing operating system, was primarily used by the same high-end, weirdo crowd that would use Lisp. PDP-10's were expensive, and machines like them pretty much died out after the PDP-10 disappeared—would you invest in a mainframe or minicomputer company today? These machines died out because there was no large population to sustain them.

The part of the argument about Unix being a virus is inacurate: Unix spread from the PDP-11 to the Vax, each of which was popular for other reasons. Unix was a parasite on these machines or a symbiot. Unix would have been abandoned for any better alternative, but it was the right thing at the time.

Seventh, as Unix had no rival on the PDP-11 and Vax, neither did C. C is very good for system programming, and that is a lot of what was done on those machines. C provides a good avenue into the system libraries and Unix utilities.

Eighth, C is not obviously a worse language: it has all the usual data types, modern control structure, and has a strong lineage—Algol 60, Cambridge CPL, BCPL, and B.

Ninth, there has never been a true, important worse-is-better/the-right-thing faceoff, and so no one can point to an actual example of the theory that one has greater survival characteristics than the other.

These arguments put the entire argument of worse is better into shadowy light. But the real quarrel with the paper I have is about what it teaches people. The paper states that a good strategy to adopt when trying to make things better is this:

> *...it is undersirable to go for the right thing first; better to get half of the right thing available so it spreads like a virus. Once people are hooked, take the time to improve it to 90% of the right thing.*

This advice is corrosive. It warps the minds of youth. It is never a good idea to intentionally aim for anything less than the best, though one might have to compromise in order to succeed. Maybe Richard means one should aim high but make sure you shoot—sadly he didn't say that. He said "worse is better," and though it might be an attractive, mind-grabbing headline seducing people into reading his paper, it teaches the wrong lesson—a lesson he may not intend, or a lesson poorly stated. I know he can say the right thing, and I wish he had.