Back to the Future: Is Worse (Still) Better?

Richard P. Gabriel

Spring-Watching Pavillion

A gentle spring evening arrives
airily, unclouded by worldly dust.
Three times the bell tolls echoes like a wave.
We see heaven upside down in sad puddles.
Love's vast sea cannot be emptied.
And springs of grace flow easily everywhere.
Where is nirvana?
Nirvana is here, nine times out of ten.

—from the Vietnamese of Ho Xuan Huong

Technology, art, popular media including network TV, and just about every aspect of our lives and probably life itself follows a disappointing pattern: worse is better, the good drives out the excellent, and the most popular is least good.

You can look at it like this, perhaps: To appeal to the majority of people, an artifact must appeal to something that those people have in common; as we increase the set of people we consider, the less those people have in common, and that which they do have in common becomes more base. For example, network TV is flooded with adolescent humor, sex, and violence because these are the most basic drives people have in common, while an interest in Vietnamese poetry—which is a very refined poetry—is quite rare. I call this the ***intersection effect***.

More surprising, the phenomenon of worse is better occurs at all levels of scale, so that even within a relatively esoteric subfield like object-oriented programming and languages—a subfield of programming languages within the subfield of software within the fields of computer science and computer engineering—we see that the good drives out the excellent and worse is still better: C++ is still the language of choice though its kissing cousin, Java, is gaining popularity. CLOS (an advanced OO language with metaobjects and tremendous power), Smalltalk (perhaps the purest form of OO), Eiffel (well-thought-out and elegant), and Self (simplicity embodied) all sit on the sidelines while all the starters are C++ and Java.

We see the same thing in other languages. Common Lisp is a wonderful, dynamic language which happens to include CLOS. It runs roughly the speed of Java or better, its runtime is smaller than Java by a lot, it has a programmatically portable executable format for code, and yet it is not only not popular, but it is not even taken seriously as a programming language by just about everyone. As we look at Web programming where version skew is a key problem, where statically defined interfaces cannot really work well—as common sense tells us— and where intelligent agents make a lot of sense, Lisp is not even on the junior varsity. Never mind that Yahoo Store and parts of Orbitz are written in Common Lisp and that NASA's Deep Space 1 space probe was written in Common Lisp.

This second example is illustrative of a devastating point: It is not simply that worse programming languages prevail through a reduction of quality by the intersection effect, but the perpetuation of worse programming languages, once they become popular, are argued for and acted on stridently. All planning for running experiments and autonomously navigating the Deep Space 1 probe is done using Common Lisp code executing on the spacecraft. It is possible for controllers to bring up a Lisp prompt on Earth and to debug and patch running code somewhere in space. Last year, this Common Lisp code was selected by a NASA panel for NASA's software of the year award. Despite this and despite the fact that the software works well in space, one of the high officials at NASA blocked the award and declared that it would not be given unless the system were re-coded in C, in which language it would be obviously better because . . . um, because . . . ?

We have seen the same thing happen to Prolog, Smalltalk, Self, ML, and Haskell (despite the rumors of a Microsoft Haskell environment). Speaking of Microsoft . . . .

More disappointing is witnessing this same effect at work in the patterns community. Christopher Alexander's ideas of patterns has as its smallest part the pattern form—the concept of patterns really has to do with pattern languages and QWAN (the Quality Without a Name). It is not about construction tricks. It is about building artifacts not only suitable for human habitation, but artifacts that increase their human inhabitants' feeling of life and wholeness. Alexander is all about beauty and quality, not about how to stick things together cleverly.

Yet, the most popular form of software patterns is exemplified by those found in "Design Patterns," by Gamma, Helm, Johnson, and Vlissides, which contains little more than techniques for coding in C++ constructs found in other programming languages—for example, 16 of the 23 patterns represent constructs found in the Common Lisp language. There is no pattern language involved, and there is nothing about QWAN. Interest in patterns is coagulating around the so-called Gang of Four style, and it looks like things could get worse. In fact, I would say that patterns are alive and well as a form of documentation and a quest for clever solutions to common programming problems, and pattern languages, QWAN, and the quest for a better future are now on their way to the sewage treatment plant—the same place they went to in the world of architecture. Down with quality, up with clever hacks. Why worry about what makes a user interface beautiful and usable when you can wonder how to do **`mapcar`** in C++.

How could this come to be? Are there good reasons—like it is better to release something initially that is not so good but on the right track and then let a community of inhabitants repair it using piecemeal growth? Or maybe it's that lower cost, otherwise less effective technologies eventually push out overpriced and over-engineered competitors? Or is it that quality is like Vietnamese poetry and thus rarely appreciated? Is it really the statement of the base nature of human taste?

Read the Vietnamese poem at the start of this piece; didn't you like it?