Back to the Future: Worse (Still) is Better!

Richard P. Gabriel

Many people find the worse-is-better philosophy rather funny, more of a parody than anything else. It is a fascinating philosophy because it sounds silly while at the same time it works. The problem with ***The Right Thing***—the anti-worse-is-better philosophy that most people tout as best—is that it can work only if luck is on your side, and then rarely even when it is.

In this position paper I will focus on programming languages, but the argument works in many other contexts.

Alexander explained it best when he talked about how living, whole, QWANful designs are ones that are developed over time by their inhabitants while paying attention to the smallest details. He talks about details down to 1/8" as being the scale you need to pay attention to in something the size of a house. In houses, to work at that level you need appropriate tools—tools that can work at that level of scale comfortably. Modular parts don't work because they have set those details in ways you cannot modify.

Many high-level programming languages hide detail by providing big abstractions. This means that you are working with modular parts and not with small pieces constructed by hand. For the right level of close detail you need a low-level language like C, assembler, or C++. With these languages you can control data layout and design very precise and exact algorithms. With Lisp and Smalltalk, for example, you are working with their big abstractions in a ham-fisted way—just fine for prototyping and understanding larger issues, but not good for minute algorithm and data structure design. For these we need craftsmen, not storyboard designers.

The only problem that the so-called high-level languages solve is being able to put large things together quickly.—this produces mock ups. The real problem is thinking that you can build whole, alive software fast. Alexander never says great buildings are built quickly; in fact if anything he says the opposite. You need to built, inhabit, feel, then build some more, over time.

Habitable software means nice interfaces and all that, but it also means a reasonable size for the software and good performance. Size and performance mean a lot to usability, and for these you need control. The ability to work at small scale and to control size and performance comes from low-level languages. It does not hurt that programming in such languages is difficult, because there is merit in going slowly.

I get a little impatient with people who confuse Alexander's concepts of aliveness, wholeness, and QWAN with someone's aesthetic: Face it, languages like Lisp and Smalltalk are just someone's idea of elegance and prettiness. They might have some wholeness and aliveness to them, but that does not mean they can be used to build things that are alive and whole. They are themselves modular parts.

Come on. Alexander says, pay attention to the details and do things slowly according to feelings, not according to abstract aesthetics.