Languages as Architecture

I found the analogy between architecture and programming languages in the January/February issue of Internet Computing very interesting (“Post-Javaism,” by Andrew P. Black, vol. 8, no. 1, 2004, pp. 94–96). Having worked as both a computer programmer and a building architect, I’d like to add some of my own observations.

Architects tend to craft buildings as one-off artifacts. Modular reuse is uncommon beyond basic materials and details such as windows, standard wall assemblies, and so on. To solve specific design problems, architects draw details differently for each building. Similarly in programming, C-derived languages remain popular and allow access to low levels of detail, probably because programmers feel that high-level modular or otherwise restrictive program-generation systems wouldn’t give them sufficient control over system design. As Black’s article points out, Java takes a step toward standardization and reuse by providing many classes for common tasks, but it also allows micromanipulation by providing primitive arrays and simple control structures.

Similarly, in construction, the architect’s drawings show general interfaces to structure and mechanical systems, and the appropriate engineers append mechanical and structural drawings.

Unix and its derivatives provide a tools-based approach to programming, but standardization beyond this has not really happened yet. Programmers persist in using shells out of the box or through scripts instead of building radically new higher-level ones. Many users still use xterm when they could have much richer views of data and files (as databases, for example, or as associative data). There’s a tendency to want to craft what happens at the lowest possible level, but there’s no reason we can’t have both high- and low-level standard models and layers. It just means we might have to revisit the application/OS distinction and move the boundary up.

In today’s world of legacy systems and “urban sprawl,” programmers still feel the need for the equivalent of shovels and blowtorches as well as prefabricated modular concrete elements. Much higher-level language components and standards must emerge, as they have in hardware design, but given the monopoly situation in the OS world, narrow interests could dictate design choices. You could argue that this has already happened in the GUI world.

Nevertheless, to progress, I believe that language research should encompass operating environments and their interfaces and migrate the body of knowledge accumulated into the languages beyond the current class libraries. To do this, we might need a more integrated and high-level approach to writing programs, one that strongly abstracts from the history of system (engineering) and application (craftsmanship) software realms.

—Tony Barake
Brockville, Ontario

Andrew P. Black responds:
I was interested to read Tony Barake’s comments; there are indeed many parallels between architecture and programming, as well as some striking differences.

One of the most significant differences is that when an architectural design is constructed, people are involved in every step of the process. As Black’s article points out, Java takes a step toward standardization and reuse by providing many classes for common tasks, but it also allows micromanipulation by providing primitive arrays and simple control structures.

It’s the apparent ease and prevalence of detailed craftsmanship that distinguishes architecture and programming from established engineering disciplines. Programming is a craft — for better or for worse — and as an engineer, I sometimes wish it were less so. In programming, craftsmanship meets engineering at the OS interface.
details. In other words, programming is design, not construction, but design that goes down to an extreme level of detail. Construction is essentially free.¹

This difference in cost structures is one of the reasons that programmers are more likely to design special-purpose string packages than architects to design special-purpose windows. Although the design costs might be similar, the manufacturing costs in the first case are zero, whereas in the second, they are so enormous that only the largest special-purpose buildings can justify them.

Although the appearance of some buildings is only skin-deep, this is certainly not true everywhere. Indeed, many architects would argue that in truly great buildings, the structural system is the architecture. It might be possible to give the Seagram building a different veneer, for example, but the striking contraposition of the almost-empty plaza and the monument qualifies it. To take another example, the most striking feature of many a building is its fenestration — the size, placement, and spacing of its windows and doors. Because this also constrains the nature and placement of structural members, fenestration and structural design are often closely intertwined. Mud huts, yurts, post-and-beam barns, and steel-frame office buildings are what they are because of their structural systems.

Although we might be better off if programming were more like engineering, my original article was about language design. Programming language design is not and should not be engineering; it is art. The analogy with architecture is again strong — both architecture and language design must pay attention to engineering constraints — but within these constraints, their job is to create a space that people love to inhabit.

I very much appreciated Barake’s closing plea that programming languages and operating systems be better integrated. One of my favorite observations on the latter comes from Dan Ingalls, who wrote, “An operating system is a collection of things that don’t fit into a language. There shouldn’t be one.”² The fact that the integration of language and system is difficult — and far more so today in the world of internetworked heterogeneous computers than when Ingalls was writing — should remind us how important it is for programming-language and operating-systems people to be cross-trained in each other’s disciplines. The interdisciplinary walls that we initially erected for our own comfort are perhaps turning into a prison.

References