

Another View of Software Design
Mitchell Kapor, mitch@kapor.com
October 6, 2003
NOT FOR CITATION

Design means vastly different things to different people. I was again reminded of this by seeing the phrase "science of design" figuring centrally in the advance materials for the conference.

These notes reflect my understanding of software design and some of the issues which I think arise as we consider it in the context of open source activities.

To me, design is, among other things, a way of organizing understanding and activity which is complementary to science (and engineering). In this frame, the "science of design" is a non sequitur. I therefore understand I am using it in a very different way than the term is being used in the conference, which I see has to do with design as the general underlying plan or approach to making piece of software (a class of undertaking which I would certainly agree is worthy of much greater systematization).

I'd like to share my particular angle on software design, something I've been advocating since 1990, and start a conversation between the two senses of design, my own and the one espoused in the phrase "the science of design". See, for instance, "The Software Design Manifesto", at http://www.kapor.com/homepages/mkapor/Software_Design_Manifesto.html.

At the Open Source Applications Foundation, a major experiment in adapting good software practice to an open source development methodology, is underway. I hope the opportunity to share some of what is going on at the conference. My weblog <http://blogs.osafoundation.org/mitch> is my ongoing record of this experiment.

My notion of software design is that it is an activity and body of understanding akin to other design disciplines such as graphic design, industrial design, and architecture (of buildings) which mediates between the world of inanimate stuff (atoms or bits or both) and the world of human needs and purposes, crafting artifacts from the former to serve the latter.

Software design and engineering complement each other, in the same way that architecture and mechanical engineering come together in construction projects.

Software design relies on science, and, in particular, on research in the area of computer-human interaction (CHI) to provide a foundation of knowledge which informs the practice of the craft of design.

Good software requires not only excellent design but excellence in engineering as well. A beautiful house with a leaky roof fails to meet human needs as much as a program with a pretty interface which is riddled with bugs.

On the other hand, living in a house even if the roof has been fixed can still be a dismal experience if proper attention has not been paid to how people actually prefer to live (cf. Microsoft Windows).

The first architecture critic whose work survives, the ancient Roman Vitruvius said good buildings exhibit the qualities of firmness, commodity, and delight. The same might be said of good software. Just as a building should sit firmly on its foundation, good software should be built on a robust foundation and be free of bugs. Commodity refers to usefulness, a quality of good builds and software. Usefulness, of course, implies a purpose for the use, and that purpose is always defined with respect to the world of human needs. Kitchens are for cooking, among other things, and word processors are for creating documents. Delight is that experience which results when all of the parts and pieces come together in a satisfying experience of and with the place or tool.

Despite it being a typical practice in commercial software development, user interfaces created "after the fact" succeed poorly. The power is with the developers, not the designers, but if design of the user experience is not integral to development, the result is usually unhappy.

The greatest successes of open source to date have been in the area of operating systems (Linux), server software (Apache), languages (Perl, Python), and other system software (GCC), not in applications. Open source has progressed to a significant degree by programmers scratching their own itches. Amen to that!

However, this does not bode well for the role of open source in application software. Of the two most prominent open source applications to date, Open Office is closely based on Microsoft Office and the Mozilla browser aspires to be no more than a standards-compliant browser (a worthy goal to be sure, but not one which has aspirations in the dimension of good design for users). How can the community of participants in open source efforts expand to include programmers, designers, and practitioners of other disciplines (to say nothing of accommodating user-centered design!)

Open source has its roots in the voluntary contributions of programmers. In recent times, major corporations and now even foundations have found their own reasons to commit employees and funds to work on open source projects. But good design is more expensive, in the short run at least, than no design. Where will the additional funds for good design of open source projects come from?

Software design, in the sense I use it, has failed to make much in the way of inroads to date as a recognized, valued practice. In an era in which open source development takes its place as an extremely important methodology in the world of information technology, increasing the relevance and contribution of software design, as I construe it, remains a worthwhile goal with very uncertain prospects.