

The Science of Design

Richard P. Gabriel

Design is the thinking done before building something.

—Richard P. Gabriel PhD MFA

It is possible to design a house, an engine, a snowboard, an airplane wing, a user interface, a chair, a road, a chemical process, a genetic alteration, a teapot, textiles, sculpture, illustrations, a table lamp, apparel, an operating system, jewelry, and a wingnut. The process one goes through for each of these is different, but in most cases it resembles thinking and perhaps drawing. Sometimes it isn't possible to say what the design is until the thing is built, in which case the design process is more like description, and the "designer" like a mapmaker.

Design: 1a. a plan or scheme conceived in the mind and intended for subsequent execution; the preliminary conception of an idea that is to be carried into effect by action ... 7a. the combination of artistic details or architectural features which go to make up a statue, building, etc.; the artistic idea as executed

—Oxford English Dictionary, Second Edition

Designers produce designs, one way or another. Sometimes a drawing is made before construction starts. sometimes the design needs to be extracted from the built artifact, and sometimes its best form is the built thing itself. But in most cases, the process of creating the design does not resemble the structure of the design itself—because the process is a combination of creativity, analysis, reasoning, and revision interspersed with construction. Moreover, a design can be "good" or "beautiful" or "elegant" aside from an assessment of how well it fits its function—therefore, form plays a role.

Educating software designers is not done the same way other sorts of designers are educated. Design is typically taught by practice while reflecting under the tutelage of a master designer, along with a heavy dose of critical reflection on works in the design area.

Software people generally have taken seriously software design's connection to science and engineering, but rarely if ever to art. There is no primary literature of software source, neither at the code, design, nor architectural levels, there is no critical literature of software-related design, there are no schools nor curricula aimed at teaching or training in software design, there are few or no masterpieces, few or no masters, nor even a way to talk about design qua design in software. The "art" that might be out there is largely company confidential. It is difficult to make a design discipline—let alone a science—out of a practice where very few designs are visible or viewable.

The only systematic attempts I know of to capture techniques and practices of software design have been made by the software patterns community, whose members have proceeded by attempting to create a second-order literature by mining common patterns of practice and expressing them in natural language and simple diagrams. The software patterns community has been working at this for about 10 years now, largely outside academic institutions. The software patterns community has its own conferences, workshops, publications, editors, reviewers, practices, and literature.

From these observations I conclude that computer scientists and software researchers have work to do before they are as knowledgeable about and as capable at software design as they could be.

Sometimes shallow thinking is the best thinking. I believe the following describes some things we need to do to understand software design properly:

- start collecting a literature of software and software-intense artifacts so that we can begin learning what software design is about
- encourage a literature and tradition of software criticism and software design theory that goes beyond software reviews on one hand and speculation on the other
- find good designers—know them by their works—and study what they do; anthropologists know how to do this
- start teaching software design and construction seriously and rigorously (in 1998 I received an Master of Fine Arts (MFA) in creative writing and experienced how effective such an education can be—it was more rigorous, more difficult, and involved more practice by far than any MS in Software Engineering program I have heard about; visit <http://www.dreamsongs.com/MFASoftware.html> for a proposal about this)
- engage the software patterns community which is already trying to capture in a precise and principled way what good design is and how to do it
- don't confuse design and engineering
- do not abandon the art, craft, and skill of design in an attempt to make it the object of science

My message to NSF is simple: Please proceed with an open mind and an attitude of inclusiveness and diversity. Fund many small things, and resist proclaiming a new science until we've explored more thoroughly the scope of design practice in software.

Let's go fast, but let's not forget what we are after.



Richard P. Gabriel received a PhD in Computer Science from Stanford University in 1981, and an MFA in Poetry from Warren Wilson College in 1998. He has been a researcher at Stanford University, President and Chief Technical Officer at Lucid, Inc., Vice President of Development at ParcPlace-Digitalk, a management consultant for several startups and Sun Microsystems, and Consulting Professor of Computer Science at Stanford University.

He currently is a Distinguished Engineer and chief scientist of a small laboratory at Sun Microsystems, researching the architecture, design, and implementation of extraordinarily large systems as well as development techniques for building them. He is Sun's open source expert, advising the company on community-based strategies. He is also President of the Hillside Group, a nonprofit that nurtures the software patterns community by holding conferences, publishing books, and awarding scholarships.

He helped design and implement a variety of dialects of Lisp. He is author of three books ("Performance and Evaluation of Lisp Systems," MIT Press, "Patterns of Software: Tales from the Software Community," Oxford University Press, and "Writers' Workshops and the Work of Making Things," Addison-Wesley Press), with two forthcoming in 2003–2004 ("Innovation Happens Elsewhere: How and Why a Company Should Participate in Open Source," (Morgan Kaufmann), and "Leaf of my Puzzled Desire," (Dreamsongs Press)), and a third ("Drive On") in manuscript. He has published numerous poems and more than 100 scientific, technical, and semi-popular papers, articles, and essays on computing.

He is the lead guitarist in a working rock 'n' roll band and a poet.