

Position Statement
NSF Workshop on Science of Design for Software-Intensive Systems

Alexander L. Wolf

University of Colorado
Boulder, Colorado USA

<http://www.cs.colorado.edu/users/alw/>

I have been perplexed by the notion of “software design” for as long as I have been a computer scientist. It has also struck me as ironic that we, as researchers, have spent more time worrying about notations and methods than about how to recognize or develop a good design. In fact, software design is no different from most approaches that software engineering researchers take to a problem: when we do not know how to solve a problem ourselves, we invent a language (or add a level of indirection or define an interface layer or . . .) that allows—or forces—someone else to solve the problem.

I have also fallen into this trap. I spent several years developing and refining the notion of “software architecture”. Research into software architectures was, and still remains today, a very important topic. But I, and most others, have gone about that research in the wrong way. We spent too much effort figuring out notations for capturing architectural designs and checking descriptions written in those notations for “correctness”, when we should have been figuring out what it means and what it takes to create a “good” design in some particular domain, not just a well-formed description of that design or domain.

Now, this is not to say that no one knows how to create good designs. Quite the contrary, given the remarkable way that software has transformed nearly every aspect of modern life. But the costs and risks of embarking on a software design are still too high: the failure rate of software-dependent projects is not in line with the failure rates of projects conducted in other critical engineering disciplines. As a gross measure of our maturity, it means that, by analogy to civil architecture, too many buildings collapse and for the same reasons.

Fundamental to an appreciation of the complexity of the problem is an understanding of what design means in an engineering discipline. Let us assume for the moment that there is indeed a software engineering discipline. William Wulf has said that engineering is not applied science, but rather it is design under constraint. What are our constraints? Traditional engineering disciplines look to nature and its physical laws. In software engineering we have the constraints given to us by the theory of computability and computational complexity, and for distributed software systems the additional constraints of bandwidth and network speed. But these constraints operate at a level of concern far from that of most problem domains. We have a hard time constructively and effectively using these constraints to guide the designs of, say, an air traffic control system or a web-based e-commerce application.

Again looking at traditional engineering disciplines we see that the goal is to master complexity, master the design process by constraining innovation. In a sense, this is the difference between a civil engineer and a civil architect. The civil engineer is taught to apply well-understood analysis techniques to well-understood combinations of well-understood building blocks to satisfy a particular technical challenge. Innovation, typically, is encouraged to be “invisible” to this process—that is, the innovation should fit within the constraints of the well understood. The civil architect, on the other hand, is taught to seek innovation in design, and is rewarded for it. The constraints that the civil architect works under are social and human centered, and are guided by aesthetics rather than physics. In situations where innovation is not required or not desired, the business case leads to the use of a civil engineer rather than an civil architect.

In software engineering we can see an emphasis on reuse in the guise of libraries and middleware frameworks. Thus, we are beginning to have reasonably well-understood combinations of reasonably well-understood building blocks, although we still lack the techniques that lead to analytically understood designs

as opposed to the more risky and higher-cost experimentally understood designs. In any case, we can expect that this part of the problem—constraining innovation in the context of true software engineering—is slowly but surely being solved.

What about the other part of the problem, that of “visible” innovation? In fact, visible innovation is a key part of the software industry, much as it is for civil architects. For us it means an emphasis on new features and new functionality. But unlike civil architects, we are not taught *how* to design *sound* innovations, and this is where many projects fail. In our discipline we do not distinguish architects from engineers in the sense that we do not specially train them for the task of disciplined innovative design. In fact, we expect to be able to harvest good architects from the ranks of good programmers. Good civil architects are not (generally speaking) good civil engineers with 20 years of experience. Is there a lesson to be learned here?

So, our discipline is at a crossroads: do we continue to exclusively develop the notion of design in the context of engineering or do we take a cue from another field and attempt to develop a new, complementary discipline of *architectural design*, one with its own selection process, curriculum, training, certification, and perhaps eventually its own academic environment? The answer to this question is critical, since the choice leads to two, very different agendas.

Biographical Statement

Alexander L. Wolf is a Professor in the Department of Computer Science at the University of Colorado, Boulder. He joined the faculty in 1992 after five years as a Member of the Technical Staff at AT&T Bell Laboratories in Murray Hill, New Jersey. While at AT&T, Dr. Wolf conducted research in the areas of large-scale distributed software systems, object-oriented database management systems, software process measurement and evaluation, and program analysis tools.

Dr. Wolf is the director of the Software Engineering Research Laboratory in the Department of Computer Science. He is also founding director of the Computer and Communications Security Research and Education Center in the College of Engineering and Applied Science, and a faculty affiliate of the Science and Technology Policy Research Center.

Dr. Wolf’s research interests are in the discovery of principles and development of technologies to support the engineering of large, complex software systems. He has published papers in a variety of areas, including software architecture, software process, and configuration management, and most recently in the areas of security, survivability, dynamic reconfiguration, and content-based networking. These projects have resulted in the construction of several innovative prototypes in use today. Dr. Wolf has been the principal investigator on numerous grants, including those from the U.S. National Science Foundation and the U.S. Department of Defense.

Dr. Wolf received a B.A. degree in Geology and Computer Science from Queens College, City University of New York. He received the M.S. and Ph.D. degrees in Computer Science from the University of Massachusetts at Amherst.

Dr. Wolf is serving as Chair of the Association of Computing Machinery (ACM) Special Interest Group on Software Engineering (SIGSOFT), which is an international professional organization of over 3000 software engineering researchers and practitioners. He is on the executive committee of the Impact Project, which is aimed at understanding and documenting the impact of software engineering research on software engineering practice. He is serving on the editorial boards of the ACM Transactions on Software Engineering and Methodology, and the Software Process Journal. He has been a program chair of several international conferences, including the 2000 International Conference on Software Engineering held in Limerick, Ireland.