

Policy Modularity: Toward a Science of Socially-Embedded System Design

Doing Science of Design: Artifacts-as-Phenomena

My ongoing research takes an unconventional perspective on the role played by scientific methods in the field of computing. This is best understood by comparing it with the *discover-then-validate* model, where principles are discovered as a result of critical analysis and synthesis of the ideas of others. Science enters the research process after the innovation has already appeared and is used to validate and help iteratively improve the innovation through measurement-based hypothesis testing and comparisons. Discover-then-validate places great weight on the community to discover principles to validate. In the area of software engineering, it has long been known that clever innovations get adopted very slowly (Riddle, 1984; Shaw, 2001) and that an expensive and problematic entangling of research and practice is often necessary (Potts, 1993). The alternative that I am exploring, *artifacts-as-phenomena*, is to conduct scientific research into the artifacts of computing as if they were natural phenomena, incorporating into the resulting analyses data obtained from designers, administrators and users concerning their intentions and experiences. The goal is not to propose a new design methodology in the short term or even a set of useful heuristics. Instead, the goal is to develop a principled understanding of how software works.

Results of a Science of Design: What Artifacts Explain

To accomplish such a broad and ambitious objective it is necessary to adopt a specific and narrow perspective on what software is and what it means to design software. Two cultures have historically been concerned with this problem: technicians and critics. Like the two cultures of Snow's (1964) famous essay, they are concerned with the same problems ultimately, but take such different perspectives that the contributions of each are unappreciated by the other. Technicians, including software engineers, user-interface designers, security engineers, etc. are concerned with the technical integrity of the product. To them, to design is to produce a system that meets the stated requirements in a well-engineered fashion. To technicians, software is running code. Critics are those with policy, organizational or social concern (e.g. Sachs, 1995; Zuboff, 1988, among many), who frequently have complained how systems affect the ways in which their users and other affected individuals conduct their lives within and outside formal work environments. To critics, design is whatever produced the resulting software, which is not running code so much as expressed and enforced policy.

My goal is to provide a principled framework for scientific discourse between these two cultures. For the purpose of my research, software is taken to be *reified norms and policy*. Software reifies these norms and rules, because it makes rules real objects of study, separate from the contested opinions of the people who required, developed and use the software. Without being too fanciful, one can effectively ask a running program "how do you think people should act in this situation?" and, if the question is framed appropriately, get a precise answer. This may not be an answer that pleases all stakeholders, but that it is the answer is seldom in doubt. Software is usually implemented without due consideration to these normative consequences, and the code that implements norms and policies in a system can be dispersed throughout its code base. They therefore lose transparency and cannot be called into question or changed easily. The consequences for users of this opacity and inflexibility range from mild inconvenience to serious disenfranchisement or abridgments of due process. The consequences for development organizations range from inconvenient and expensive maintenance operations to major liabilities. Regarding software as reified norms and policies places the emphasis where it belongs in principled design practice: on formative design principles grounded in evaluating effects on people. The perspective advocated here promises to bridge the two cultures of technician and critic productively. It places attention on the design process so that it can be improved, as the software engineering community has always done, but it does so at the level of normatively observable effects, not at the level of software

architecture. This, the policy community has always done. Regarding software as quasi-engineered virtual devices does not emphasize the social embedding of the complex systems that sustain modern society's civic infrastructure and the consequent effects of those systems on our way of life.

Doing Science of Design: Communities of Practice

The grandiose aim is to establish a new area of research with an associated community of research investigators who currently work in the areas of human-computer interaction, software engineering, management information systems, information security, and public policy. Modeled after the patterns community within the area of object-oriented design, this community will conduct artifacts-as-phenomena investigations concerning existing systems and interfaces, abstracting the phenomena that work, fail, interact, ensure flexibility or thwart flexibility. The artifacts that I treat as phenomena are user interfaces. They have the great benefit of being as objectively present as source code (unlike, for example, the meanings left unstated between the lines of design documents), are visual or tangible in a way that source code is not, and it are more easily related to the norm or policy implications of a software system.

Further details at both the procedural and results-oriented level can be found in a recent NSF SGER award description off my home page: <http://www.cc.gatech.edu/fac/Colin.Potts/research/sgerPolicy.pdf>.

Biographical Sketch

I have worked on and off for over twenty years in the study and development of design methods (e.g. requirements and software engineering methods and design rationale) and have come to the conclusion that a science of design is more likely to be achieved and more likely to have engineering value if it is framed as a science of the designed rather than a science of designing. This is a comparatively recent epiphany, but other recent publications reveal the development of my ideas in this direction:

- (1) My studies of the evolution of telephony services (Anton & Potts, 2003) and work product software applications (Hsi & Potts, 2000) are existing efforts to study system evolution through artifacts rather than design rationale and "from the outside" rather than in terms of their architecture.
- (2) Stemming from joint research with Prof. Anton's group at North Carolina State University and funded jointly by an ITR grant (#0113792, ITR/PE: Aligning Societal Values, Privacy Policy, and IT Requirements), I have recently completed an analysis of privacy goals and vulnerabilities for Internet browsing (Jensen & Potts, 2003), the data for which is both the observable behavior of browsers (as proposed in this research), the semantics of the privacy preference language P3P and known vulnerabilities of browsers and other Internet components. These studies reveal the norms and policies (concerning one's autonomy and rights over information) that are built into software artifacts, rather than the aspirations of designers and policymakers.

References

- A.I. Anton and C. Potts. Functional Paleontology: The Evolution of User-Visible System Services, IEEE Transactions on Software Engineering, 29(2), pp. 151-166, February 2003.
- Hsi, I. and C. Potts (2000). External Complexity and Evolution of Software: Reverse Engineering without Code. Proc. Int. Conf. Software Maintenance: San Jose, CA.
- C. Potts, Software Engineering Research Revisited, IEEE Software, September 1993, pp. 19-28.
- Riddle, W.E. 'The Magic Number 18, Plus or Minus Three: A Study of Software Technology Maturation,' SIGSOFT Software Eng. Notes 9(2): 21-37, 1984
- Sachs, P. "Transforming Work: Collaboration, Learning, and Design." Comm. ACM, 38(9), September 1995. pp. 36-45.
- Shaw, Mary. The coming-of-age of software architecture research, IEEE 23rd International Conference on Software Engineering, Toronto, Ontario, pp. 657 - 664a, 12-19 May 2001.
- Snow, C.P. The Two Cultures and a Second Look: An Expanded Version of the Two Cultures and the Scientific Revolution, Cambridge University Press, 1964.
- Zuboff, S. In the Age of the Smart Machine: The Future of Work and Power, Basic Books, 1988.