Book Review (published in *Computer-Supported Cooperative Work*)

Social Thinking – Software Practice. Yvonne Dittrich, Christiane Floyd & Ralf Klischewski (eds).

Paul Dourish
University of California, Irvine

The emergence of CSCW as a distinct area of intellectual inquiry and professional practice was grounded not least in the methods and concerns of the Participatory Design (PD) movement. PD's concern for direct engagement with work settings and those who occupy them has been methodologically central to CSCW practice, and PD's central commitment to the complex of social forces that surround technology design, deployment, and adoption has been similarly adopted as a central tenet of CSCW investigations. PD continues, of course, and there is still much to be learned from the experiences of researchers working in the PD tradition. "Social Thinking – Software Practice" is a collection which is firmly rooted in this tradition, and it bears PD's hallmarks. In particular, it displays both the urgency and reflection that characterize the PD enterprise.

The book has its beginnings in a workshop held in 1999. That is a common origin for edited collections, but this is, fortunately, an uncommon collection. In particular, the editors and authors have gone to considerable lengths to attempt to integrate the material and present it as a coherent whole. Many collections that emerge from workshop proposals are "bitty" – but there is a strong sense here of the relationship between the contributions. This is partly a result of a good deal of internal reference between the papers, reflecting the process by which the contributions were produced, but also a result of the organizational strategy that has been employed. The book is divided into five sections (entitled "Deconstructing", "Informing", "Grounding", "Organizing", and "Reorienting"), each of which is anchored by an introduction that sets the tone and ties the contributions together. This serves not only to provide a coherent intellectual framework, but also to make readers wish that they had participated in the workshop from which this book emerges; the organization lends a vibrancy and a sense of an emerging conversation. One has the sense of the workshop as a coherent and fascinating event. Everyone editing a collection of workshop contributions should look to this volume as a model of how to go about the process. Indeed, I would like to have seen it taken further; there is still a good deal of repetition between the chapters, especially around motivations and historical perspectives on both Software Engineering and Participatory Design. However, this is nit-picking; the editors are to be praised for the coherence they have achieved.

One thing that strikes the reader immediately is the wealth of real-world experience on which the reflections offered here are based. The articles here are not abstract ruminations on the process of design; they are reflections and reports on real design efforts, and real experiences. The collection is valuable for that alone. In addition, though, it also offers both valuable design techniques, methodological comparisons, and thoughtful critiques which will be of value not just to PD researchers but also to many in allied areas such as HCI and CSCW.

That said, of course, any volume must leave more unsaid than said. After a couple of readings, I was left still unsure about some issues that I had imagined would be central to the book, as prefigured by the juxtaposition of the title, between social thinking and software practice.

First, what is the fundamental nature of software here? Ironically, in the book (as is not uncommon), software itself tends to disappear, to be replaced with a more generic notion of engineering design. The fact that it is software that is being designed – rather than car parts, oven mitts, or cardboard packaging -- frequently seems to be incidental; the focus is on design as a process, but often with little said about the product. With the notable exception of Christiane

Floyd's contribution early in the volume, software never comes squarely into focus. This raises two subsidiary questions in the mind of the reader. The first is, aren't there some unique properties of software systems that are worth exploring here? It is my personal conviction that there most certainly are. Software systems are representational and intentional (in the technical sense); they are systems through which people create and share meaning. Properties of agency, activity, representation, malleability and formality are ones that, I believe, make the interaction between software and human systems unlike those between people and other forms of engineering. Indeed, one might be tempted to question the notion of software engineering by demonstrating that software systems are not like other engineered artifacts – a perspective that is not strongly represented here. Partly, perhaps, that is because, while "software" is the proclaimed topic of the book, "Information Systems" is perhaps a more accurate depiction, and Information Systems are a different kind of animal altogether. However, that leads us to the second subsidiary question. If our topic is engineering design, then why is it that other forms of engineering design don't seem to require either the methodological prescriptions or the reflective concern that is exhibited here? Looking on software systems as arising from a process of engineering design not dissimilar to that which is conducted around bridges, airplanes, and ballpoint pens, one is tempted to ask what the role is for social science as an aspect of the design of bridges, airplanes and ballpoint pens. Why are they not the topic? Why are social scientists not conducting ethnographic investigations of ballpoint pen use, and organizing future workshops to engage their uses in the design process? If – as I believe is the case – this is because software has some unique properties when incorporated into working practice, then we might expect that those unique properties of software would play a central role in social analysis of software systems.

Comparisons to ballpoint pens may seem rather facetious, and I am exaggerating a little for effect, but the fundamental point is a serious one. A focus on software systems in particular requires a fundamental engagement with the nature of software, not simply with software systems as social artifacts or with design processes and social processes. Even the title of this volume attests to the central role of software in particular; but, for many of the contributions, software essentially disappears.

In turn, this leads us to a second question. As is not uncommon in the research literature, this volume contains a catalog of system failures and analyses that account for these failures by exposing the lack of engagement with social science (and, through social science, with users and contexts of use.) However, it is rather more reticent on the topic of software successes. I write this review using a successful (not necessarily good, but certainly successful) word processing system, running on top of a successful operating system, replete with other successful applications. A host of software systems will also be responsible for bringing this article to its readers; systems for typesetting, electronic communication, dissemination of electronic copy, logistics, library order systems, electronic catalogs, digital libraries, etc. It is hard to argue that the success of these systems is dependent on the participatory nature of their design processes, or their engagement with social science. How can this be? Clearly, operating systems, office productivity applications and electronic communication systems are also socially constructed artifacts. They are deployed and used in socially organized settings. They reflect a set of assumptions and expectations based frequently on minimal (at best) engagement between designers and users; and yet, despite this, they succeed. To be sure, they do not succeed to the extent typically promoted by corporate marketing or by those who hold to a technologically utopian ideal; but the gap between unremarkable everyday success and the abject failure that we often encounter in technology studies remains to be explored. This gap is present not only in this particular volume, but for us all who struggle to understand the issues that this volume cogently represents. It strikes me as an increasingly important concern. Offering symmetric accounts of success and failure is as major a challenge for us, I think, as the similar symmetric challenge that

the Strong Program in the social studies of science has issued to scientific epistemology and scientific practice.

It is hardly fair, of course, to demand that these questions be answered by this particular volume. They are, rather, collective questions for those of us engaged in research at the intersection of social science and design practice. However, they arise in the context of this book (as they do with others) because of the question of audience. Practicing software developers would quite naturally be surprised by the absence of software as a focused object of enquiry here. Similarly, many software professionals or analysts might wonder about the conception of software success and failure that are implied by our research investigations. “Social Thinking – Software Practice” provides an excellent orientation to current research directions at that intersection; but in doing so, it illuminates not only what has been done but also what remains to be tackled. Readers who seek the solution to problems at the intersection between social thinking and software practice will not find them here, because as a community we do not yet have the answers to provide. Readers who wish to understand current thinking, theoretical concerns, and methodological approaches and to find thoughtful and engaged critiques of software development practice, however, will find this collection invaluable.