Social Thinking – Software Practice

Approaches Relating Software Development, Work,
and Organizational Change

Scope of the Seminar

During the last decade, the embedding of software into work practice has stimulated interdisciplinary cooperation between social scientists and computer scientists in areas such as computer supported cooperative work (CSCW) or human computer interaction (HCI). The discussion is largely driven by experiences gained in development projects. Key questions include: How can the perspectives and conceptualizations of different users and other stakeholders be taken into account? What is the relation between, on one side, reflective and analytic abstractions of the social sciences and, on the other side, generative abstractions developed by the designers in order to be implemented in and with a computer application? A coherent understanding of these and related questions is needed in order to provide suitable methods for software development.

Background: Social Thinking ...

Approaches developed in the social sciences for understanding human learning and communication, individual and cooperative work, and the interrelation of technology with organizations, provide a starting point for dealing with the problems at stake here. Although these approaches have been developed with no specific concern for computing, several of them have been tailored to the needs of software development and use:

- *Activity theory* and developmental work research focus on the activities of individuals, portraying these activities as mediated by tools and taking place in a social context. For understanding and changing activities they rely on representations of complex activity networks.
- *Ethnographic* workplace studies are concerned with how individual or collaborative work is actually being performed, in particular, they show the use of artifacts in work practice. Through participant observation, open interviews and other techniques they aim to understand work practice from the participants' point of view.
- *Discursive approaches* start out from the different perspectives of the actors involved, including their interests, and power relations. By facilitating communication, they aim to provide equal opportunities for participation.
- *Systemic approaches* emphasize the interconnections between actors in organizations and between organizations and technology. They focus on different levels of reality construction inherent in human learning and communication, individual and cooperative work.

These approaches emerge from different, to some extent controversial, discourses in the social sciences. So far, their applicability to software development and use has largely been discussed in separate arenas.

... and Software Practice

While the discipline of software engineering is mainly concerned with the formal principles, the technical basis and the methodological support for software development, the reflection of software practice as a human activity needs to go beyond an engineering framework. Several aspects come into focus:

- Software development is a continuous *process based on human learning and communication*, in which all activities contribute to insights into the desired functionality and use of the software to be constructed.
- *Methods* and their underlying concepts are social vehicles for technical work. They embody a perspective on software development, concerning, for example, who is to be

involved, which activities need to be supported and how, and what aims should be achieved.

- Key activities of software development, such as requirements analysis, modeling and design, *relate the technical domain* of software functions and modes of human-computer-interaction *to the social world* of work and organizational change.
- Software development incorporates *organizational concepts*. The use of software products constrains collaborative work and organizational development. Thus, reifications in software systems have an impact on the potentials of organizational change.

In order to take these aspects into account, social science approaches are needed as guidance for software development in social contexts.

Seminar Aim

The seminar was arranged so as to promote a conversation about social science approaches in their relevance to software development. On one hand, approaches from the activity theoretic, ethnomethodological, discursive, and systemic schools were presented in their applicability to software development. On the other hand, mutual understanding was facilitated by identifying complementary views and methods as well as incompatibilities.

Seminar Program and Contribution of Participants

Participants had been invited to the seminar by Christiane Floyd**,** Nimal Jayaratna, Finn Kensing, and Lucy Suchman**.** Since it was intended to foster the interaction between different “schools of thought” a group of researchers from different backgrounds were entrusted with the preparation of the seminar. The members of this group were Yvonne Dittrich (Ronneby), Ralf Klischewski (Hamburg), Olav Berthelsen (Århus), Victor Kaptelinin (Umeå), Helena Karasti (Oulu), Jakob Nørbjerg (Copenhagen), Jesper Simonsen (Roskilde), Chris Westrup (Manchester), and Volker Wulf (Bonn).

After the opening on Sunday evening the course of the seminar followed the suggested ‘themes of the day’: “Making software”, “Social Thinking”, “Software as Social Change”, and Industrializing Software Development”. During the conference all participants played an active role, e.g. leading a working group, giving an introductory lecture or taking a part in discussion. Debate was enabled in plenary sessions as well as in small working groups from which the results of the discussions were presented orally and/or on wall paper. Friday morning was reserved for summing up and discussing further projects of co-operation.

All participants were expected to submit a position paper related to the seminar’s theme. Prior to the seminar, these papers had been presented on a web site accessible only by participants. Authors had the chance to submit a revised version of their position paper to be included in this report. Following the seminar a book will be published, and all participants have been invited to submit a chapter based on their position paper.

Christiane Floyd, Yvonne Dittrich, Ralf Klischewski
December 1999

Table of Contents

An Analytical Framework for Understanding the Intertwining of Social and Technical Aspects in Software Development Projects

Urs Andelfinger

Software development projects are normally supposed to follow some sort of what is commonly called a *Life Cycle*. In Software-Engineering therefore exists a wealth of different lifecycles, e.g linear Waterfall-Models as well as various forms of Spiral-Models that are more appropriate for the evolutionary and incremental nature of software projects. Although these models are fundamentally different to some extent, we can abstract some generic commonality on a very basic level:

At the outset they start from a normally rather fuzzy *Setting of Use* for the software-product that is to be developed. The next major cornerstone is therefore reached when *the* resp. *a Requirements Specification* and *State of Shared Understanding* of what exactly has to be developed is established. Based on this specification the *Technical Implementation* is carried out before eventually introducing the software-product into its setting of use. Of course, to accomplish these tasks, *Resources* have to be allocated to the project, which cover technical methods as well as staffing, project-management know-how and funding.

This generic and rather technical perspective on software projects is widely accepted as the essence of Software-Engineering. Because this perspective is so wide-spread and because it covers the most obvious characteristic of software projects, namely their product-orientation, I suggest to call the triple *Setting of Use, Requirements Specification* and *Technical Implementation* the *Foreground* of software projects. The *Resources* are conceived of in this perspective as being means to get the product done.

When contrasting this view on software projects with practical project experience, we quickly discover that there are some important aspects missing in order to adequately understand what software projects are all about:

In real-life projects, we have to cope with *Actors* and *Stakeholders*. Also, software projects normally have technical as well as economical *Objectives* and are supposed to meet some stakeholders' *Interests*. Last but not least software projects are always embedded in a *Development Context* which comprises e.g. the general organizational culture and the politics that are inevitably linked with all software projects.

Compared to the product-oriented foreground of software projects, these aspects here are often more fuzzy, hidden and to a great extent implicit. This is why I suggest to call them the *Background* of software projects. This does not all mean that these aspects are less important than the foreground of software projects - quite to the contrary, they correspond even to the most important 'soft factors' in Management Science. The term simply alludes to the fact that these aspects are normally more implicitly dealt with in software projects.

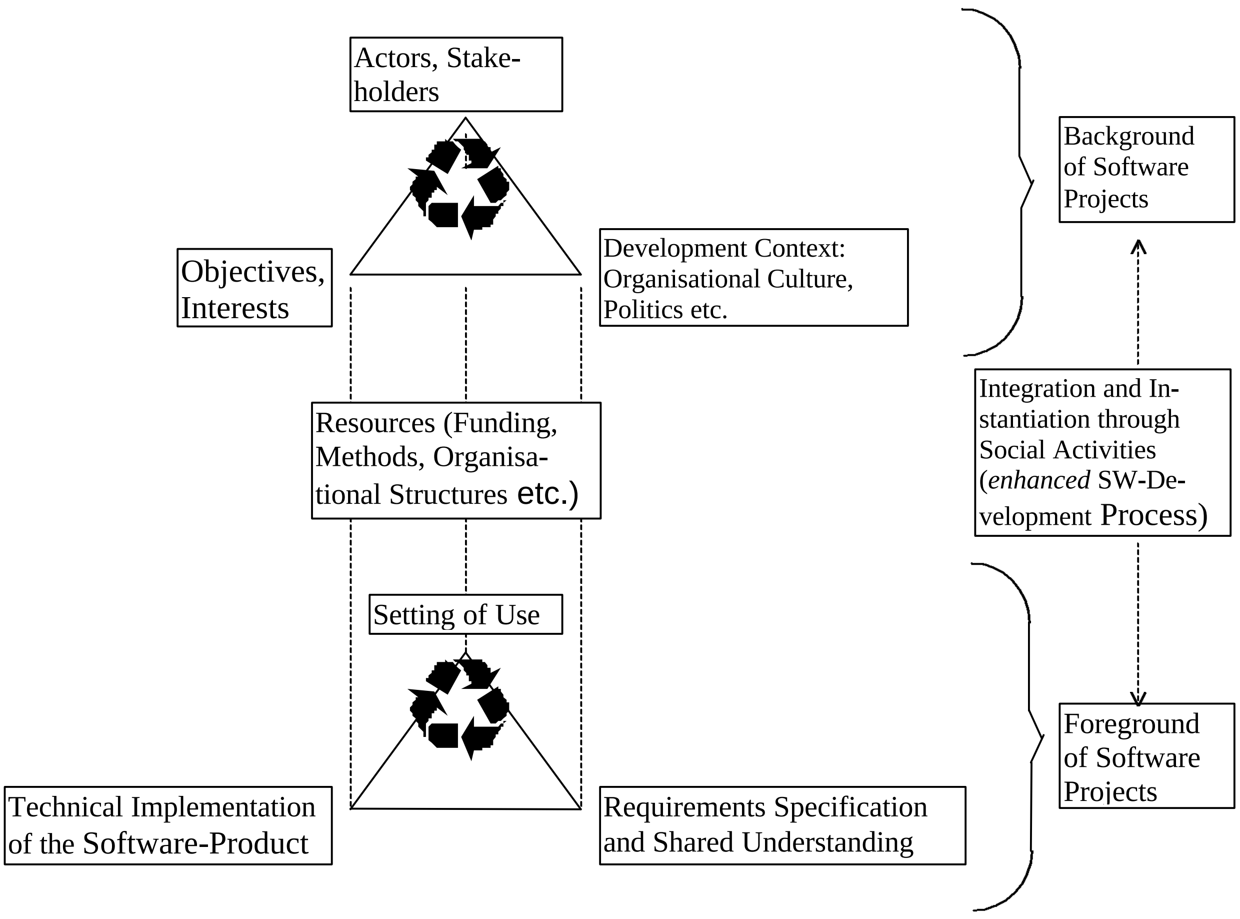
The interesting question now is how the foreground and the background of software projects are intertwined resp. related. Is there some basic mechanism? This is the point where I suggest to introduce the notion of *Social Activities*: It is by *social activities* that the foreground and the background of software projects become intertwined and - as a result - the technical work of the foreground in software projects gets done *and* gets its meaning.

One might now of course argue, that what we call *social activities* is already covered by what is commonly called *development process* in Software-Engineering: Normally, the development process in Software-Engineering is conceived of as a coherent set of 'Specification-, Design-, Implemenation-, Integration- and Testing-' activities which mainly refers to the product-oriented foreground of software projects.

Social activities extend this notion by including all forms of explicit and implicit, formal and informal activities, negotiations and technical work, be it on an individual or on a group level that help accomplish the product-oriented tasks *and* the constitution and creation of meaning. In other words, we are embedding the technically focused activities of a 'normal' development process into a broader range of activities that equally cover technical (product-focus) and social (meaning etc.).aspects.

Now, as we have introduced all components of our analytical framework, we are able to put together the whole picture of what software projects are all about:

Conceptually, they consist of a more technical oriented foreground and background that are tied together via social activities on various kinds of resources. The social activities of course produce some technical product but they are equally about relating the technical work to its background, to give it meaning and to satisfy the background's interests. The whole analytical framework can be visualised in a very simplified way as follows:



Actors, Stake-holders
Background of Software Projects
Objectives, Interests
Development Context: Organisational Culture, Politics etc.
Integration and Instantiation through Social Activities (*enhanced* SW-Development Process)
Resources (Funding, Methods, Organisational Structures etc.)
Setting of Use
Foreground of Software Projects
Technical Implementation of the Software-Product
Requirements Specification and Shared Understanding

The Analytical Framework of the Intertwining of Social and Technical Aspects in Software-Development Projects

To summarise, the key insights of the analytical framework can be described as follows:

- It offers an adequate conceptual framework for the notion of software projects as a coherent socio-technical endeavour (though quite analytic and of course very simplified).
- It suggests that software projects are about creating a technical product as well as about creating meaning and shared understanding. Particularly, it analyses the final software product as the result of the ongoing intertwining of social and technical aspects during the development process.
- It offers an explanation why communicative methods, technologies and social skills are as crucial as technical skills in software projects and it gives hints when and how to

apply each of them. In this sense it does not make Software-Engineering obsolete but it gives it a broader frame which may help to get *valid* software that meets both technical and social criteria.

Strong Theory, Strong Problems

James M. Nyce
Gail Bader

Anderson (1) describes the design and development process (and community) as bounded by a preoccupation with 'problem and solution'. In other words, the design and development process is seen as a relatively straight forward problem in information engineering and one for which a more or less 'elegant' solution can be arrived at. The design and development process, for those who take part in it, represents a 'scientific' problem that can be solved using methodical information gathering and processing procedures.

For designers and developers understanding design and development as a scientific, engineering problem is taken as a natural order of things (and this reflects how computer science is taught at university and college). The result is what designers and developers do is first order kind of work. In other words, critical inquiry is reduced to questions of 'how to do it better' and answers to these questions rely more on common sense and introspection than anything else. To give but one example, this is where developers and designers involved in instructional technology derive their understanding of what students 'are like' and what they 'need' in the classroom. To put it another way, the 'problem and solution' engineering paradigm provides the ontological and epistemological framework that both frames and informs the design and development process.

When software development gets framed this way it leads members of this community to think about and to use ethnography merely as a means to 'solve problems'. The result is what a 'problem' is and how to 'solve' it get reduced to a series of practical interventions and practical outcomes. Because of this, ethnography as the design and development community sees it, is to provide 'usable' information at essential points in the design and development process (practical interventions) that will allow 'better' more 'elegant' systems and applications to emerge (practical outcomes).Anderson (1) suggests that what ethnography can bring to the design and development community is something more valuable than 'the detailed description of work routines and daily life with which to fix the features of the design' (170). While ethnography can make (and has made) a number of contributions of this kind, Anderson suggests (and we concur) that ethnography can bring to the surface, make evident and challenge 'the taken for granted assumptions embedded in the conventional problem-solution framework' (170).

The task then is to help designers (and those 'purchasing' systems) become aware of the framework in which they work. What upholds the problem/solution paradigm? Why is it taken to be 'natural' and self-evident? What we are calling for here is an ethnography that looks at what goes on in the design and development process itself. The question that interests us is why 'work', 'process' and 'product' get constructed the way it does in the design and development community.

The social relations of the design and development process can be productively explored using ethnography. As with basic assumptions, we are very often unaware of (because we are so immersed in) relations of power and authority. These kinds of questions, while acknowledged, are not much explored. Who is the end-user and what is the end-users relationship to the 'client' or the person/representative of an organization paying for the process? Are these two the same?

Who has a stake in maintaining or 're-Engineering' work processes? Ethnography can help all parties to the design and development process look at and reflect on the world they take for granted as participants.

From this line of inquiry we can take on questions like what consequences does the problem-solution paradigm have for designers/developers and their clients? What gets masked and disappeared when this paradigm is taken to be the natural order of things? We would argue that this is why (and how) things like power, autonomy, culture, hierarchy and money get muted, masked or disappeared in the doing of design and development. In particular, we would like to argue that we need take a close look at the foundational categories ˆ ones that both uphold and have received little acknowledgement within the problem-solution paradigm.

To illustrate how ethnography can help answer questions about what underlies particular design and development agendas (see also 2, 3), we will look at participatory design and argue that it rests and 'works' on the basis of some fundamental Scandinavian assumptions (4). Among these have to do with rationality, speech and action. Participatory design assumes that group work and rational discussion are essential for design and development because it is from them that mutual understanding, consensus and agreement among stakeholders emerges. Consensus, agreement (or discussion by all 'stakeholders' in the work situation) results in both an adequate understanding of local work processes and an adequate design and development cycle.

Participatory design then rests on a set of Scandinavian assumptions about language and speech. For example, in Sweden speech can triumph over differences in power, authority and hierarchy. Because all stakeholders are brought to the table, allowed to voice their own positions and perspectives, it is believed that inequalities in status, power and knowledge can be banished or, at least, overcome though 'rational' discussion. Indeed, the solutions that emerge from these discussions are, by definition, 'rational'. That some members have more expertise or power than others is denied because each stakeholder had his or her say.

This makes little sense in the United States. First off, the idea that speech and discussion can lead to any change at all would be regarded with skepticism. What is effective and efficient, when it comes to change in US terms, is action, not speech. In fact speech has little or nothing to do, in US terms, with agency and change. Folk sayings (Show me you,re a man for example) make this point over and over again.

The idea that speech and discussion necessarily have anything to do with rationalism is equally suspect. Persuasion in American terms, while it may make use of speech, is seldom dependent on (or seen as a voicing of) rationality. Persuasion and argument are seen as having more in common with what salesmen 'will do' to make a sale (think of the role the used car salesmen has in America). Unless proven otherwise, speech and argument are seen as having little to do with rational expression.

Then there is the problem of self-interest. Almost any discussion in the US will invariably turn to a discussion (and accusations) of self-interest. In short, far from overcoming differences and inequality, discussion and negotiation for participants in the US is seen as horse trading and making deals that entitle some and leave others out. In the US, having one,s say may not matter.

What is privileged in the US is any kind of technical discourse. These are not privileged in and of themselves (for their words or rhetoric). They are privileged for two reasons. They offer the possibility, Americans believe, of strong, useful revelations about the self and what goes on in the world. In effect, they are privileged because they rest on and embody what Americans take to be scientific knowledge. Scientific knowledge is typically thought of as knowledge that has strong claims to 'truth'. This kind of knowledge is taken to be more impartial and fair as it reflects not special interests but the nature of the world. As a result, in the US, technical discourse stands the best chance of being taking seriously. In American terms, this is about as close as it ever gets to rational statement.

Claims to 'rationality' play a powerful role in design and development as it is enacted in the US. Workers who refuse to use information technologies 'correctly' (i.e., as they were designed by software engineers) or question the wisdom of automating particular work processes are labeled 'irrational'. Struggles between software developers and end users can either crush end users (and their vision of the technology under development) or devolve into clashes between different kinds of expertise (e.g., professional competencies like those of teachers and doctors vs. programming (technical) competence). Design and development efforts often take the form in the US of negotiating who can make the strongest rhetorical claims about 'solution', 'rationality' and 'technical' competence. Given that in the US, development and design efforts focus down almost immediately to issues and questions of technology, it is not surprising that end users, no matter how competent, generally come off second best (for a participatory design effort in the US where this occurred, see Blomberg [5]).

In both Scandinavian and the US, we want to develop software that will be useful and support work processes. Yet, we have very different ways of attempting to create a 'level playing field' one that can overcome differences in hierarchy, authority and power so that both structure and knowledge can emerge that will enable software of this kind to be built and accepted.

To build applications of this kind, participatory design assumes a level playing field is necessary, that this can be done and that to do this is a comparatively simple event. From a Scandinavian point of view, this at best requires only some language work. At worst, it will requires a design and development effort to come to some agreement about what principles in this linguistic exchange will 'guarantee' participant parity and equity. In the US, the way language is viewed, i.e., it represents a weak, if not suspect, form of action, and the strong, privileged role technical discourse has in the States throws into doubt whether participatory-design can level the playing field. In turn this raises the question of whether participatory-design in the US can provide either the structure or knowledge necessary to build useful and acceptable applications for the workplace.

We developed this example to show why it is important to have ethnography to look at fundamental categories that underlie the development process. As our example has shown, the design and development process is a set of social relationships that can be understood in substantially different ways based on culture (as well as power, hierarchy and authority). Not only do development and design goals differ but how we go after these goals, the evidence we use, and the appeals we make, reject and accept can differ as well.

If the design and development community was more aware of what in its own terms defined 'legitimate' action, method and thought, it would be possible to make design and development choices on the basis of something other than 'common sense'. The kind of ethnography we are calling for here would help designers and developers become aware of what they take to be self evident in their own work and practice. In turn, this would help us all make more reflective choices about the technology we need and how we should build it.

1. Anderson, R. J. (1994). Representations and Requirements: The Value of Ethnography in System Design. Human-Computer Interaction, 9, 151-182.
2. Bader, G. and Nyce, J. M. (1998). When Only the Self is Real: Theory and Practice in the Development Community. Journal of Computer Documentation, 22(1), 5-10.
3. Forsythe, D. E. (1999). 'It,s Just a Matter of Common Sense': Ethnography as Invisible Work. Computer Supported Collaborative Work, 8, 127-145.
4. Nyce, J. M. and Lowgren, J. (1995). Towards Foundational Analysis in Human-Computer Interaction. In P. J. Thomas (Ed.), The Social and Interactional Dimensions of Human-Computer Interfaces. (pp. 36-47). New York: Cambridge University Press.

5. Blomberg, J. (1990). Reflections on Participatory Design: Lessons from the Trillium Experience. Proceedings of the Conference on Human-Computer Interaction, 353-359. New York: ACM.

Computing: an accelerating whirlwind of status quo?

Eevi E. Beck

1. Troubling experiences

I wish to highlight an area where I believe strong and perhaps disturbing challenges to computing can and need to be brought up. By "computing" I mean its common meaning in 1999, usages of computers (as opposed to the original "calculation"), and issues surrounding such uses. My question for the workshop is: *to what extent can joint work between social and computing scientists help questions to be asked, challenges to be raised, in areas that deeply affect computing and society?*

My empirical studies include collaborative writing over distance, and elderly care in a small town in Sweden, as well as a smaller study of a state agency in Norway where Lotus Notes was being used. What links these is the questions surrounding computers and the structurings of work. Or more precisely: how do usages of computers and computing, in the concrete and in various political senses, enter into (strengthen, or co-create) patterns of working, patterns of being; patterns of interest and of power?

Exploring specific situations, some patterning similarities stand out to me as more interesting than others. Strong stereotypes of what computers are for seem to be perpetuated in the practices of their use at work. Collaborative writing system prototypes, as results of the research in Europe and the US up until that point (ca. 1990), were in part based on simplistic and largely unexamined assumptions of what collaborative writing was about (Beck 1994). Fitting a rationalistic and competitive view of such processes, relationship aspects were forgotten or ignored although studies including mine pointed to these as integral.

Early use of Lotus Notes in a governmental agency in Norway seemed to me to be characterised by: 1. Immaturity of the technology and questions arising from this. E.g. dependency on overworked support staff (e.g. unclear training needs; extra package to be installed for printing databases), as well as unreliable service (frequent crashes under everyday workload). So: *Why did the agency start using it at all when so new?* 2: Complex mixes of factors seemed to influence how the various employees we interviewed related to Notes. Some showed a measure of choice about whether or how to incorporate Notes use into their work (e.g. lawyers we interviewed included on the one hand, one of the most active in shaping a database, and on the other, the employee with the most distant relationship: he chose to let a secretary type his letters). Others appeared to have, or take for themselves, no such choice (the receptionist and the archivists may have seen it as a complex, undisturbable given in their work). Some tried to or did create or alter databases. Some inspectors used it minimally (only to the extent their relationship to their manager demanded it, plus for access to a word processor). Assuming my "observations" are reasonable interpretations of what I saw and heard, and that similar ones may be made elsewhere: *How can researchers–or anyone–justly characterise such diverse relatings?* Should researchers try to conceptualise differences in relating to computing, and if so how? And 3: I found indications that the visibility to others (incl. to support staff? and to manager) of different employees as users seemed to differ along (to me) familiar lines of hierarchy. Lotus Notes, while in certain ways providing a technical potential for egalitarian patterns of use, in no way served to challenge–and thereby, I claim, went into maintaining–an existing hierarchy. (E.g. the overworked support staff installed printing for managers long before they had time to do so for others; the receptionist/switchboard operator's manager had not noticed the technology-intensity of that work although she (the manager) passed the reception many times a day.) This point is consistent with Orlikowski's conclusion from her 1992 study that *the technology itself is not sufficient* to create new patterns of work. Orlikowski explained her findings in part through an atmosphere of strong competition among colleagues at the US consultancy she studied. Without making too much of my small study, it is interesting that I found evidence of similar structurings in a workplace which may have had little or no internal competition in the reward system and in a country where egalitarianism arguably stands stronger in the "national culture" than in the US (and so, by my estimation, the step to this particular kind of work practice need not have been that great). So what Orlikowski pointed to may be stronger and more subtle processes than many have believed. What are these?

So many of us (including me) develop and sustain a highly limited set of ideas about what computers are for. All the while hailing computing as revolutionary. Seeing their plasticity, some claim they can be used for "anything." Yet the combined imaginations of those who in various ways shape the development of computer applications would appear to have failed to provide much beyond that which helps someone make more money (see Donna Haraway 1997 for an eloquent examination of a number of issues relating to this). How is it so?

1A. Questions such as what power relations computer usage often strengthens, what ways of working, living, and loving (yes, loving), and to whose benefit, are in my opinion among the most essential one can ask of "computing" today. ("Essential," as in: "the essence"–or one of the "essences"–of what computers and computing do, of "what's going on.") I fiercely believe that research in computing encompasses these questions, whether or not acknowledged to do so. However, this may not appear to be the case when looking at the work going on in the four-six computing departments I know (in three countries). My argument, therefore, requires another step: that computers/computing somehow in significant ways enter into processes of major change and non-change that are not such discussed. Re. "non-change": It might be an interesting analytic turn to examine a resilience against certain kinds changes, a perpetuation of status quo in the mids of loudly professed (computing) "revolutions."

1B. I believe that self-examination, self-conscious but with the possibility of harnessing a culture of healthy self-critique, must be part of any mature academic discipline. (One could hope for the same also for professions, though for various reasons that road seems rockier.) I have never heard anyone claim that Computer Science in all its varieties is a mature discipline. It needs to mature, however, and desperately so, for the sake of Western culture and Computer Science as part of it. Such questioning (more than the answers it may produce) is a necessary step to maturing, and therefore, the project of making explicit these questions must be part of Computer Science.

This is where Computer Science, if it is an academic discipline, needs the support of others with a tradition of debating self-ransacking issues.

2. Rationalities of Responsibility – an anomaly to IT?

I am conducting a long-term study in a local government agency responsible for providing home help assistance in a small town in Sweden. In my paper ”Managing Diffracted
(1997) I describe some of this. Computers were used by managers and were being introduced in a pilot group of home helpers. In the paper I adopted and adapted terms of Bjørg Aase Sørensen's

(1982) to make a somewhat different point than her original one. Sørensen wrote in 1982 of a rationality of techno-economics which was different from a rationality of responsibility (and varied along gender lines in the factory case she explores in her paper). To explore aspects of a case of IT use in a local government elderly care unit, I separated the terms rationalities of economics from those of technology. I argued that while there may be a descriptive truth in the the intimate linking of technologies (including IT) with rationalities of economics, this need not be so. And this difference is important.

Intersections between IT and what I for convenience loosely term rationalities of responsibility is an example of what I wish to highlight as more or less wholly overlooked areas for exploration.

3. Change

Computing technology itself has not revolutionised work life (except in highly specific senses). As long as our ways of using it is governed by traditional ideas of what work is, technology will continue to reinforce traditional hierarchies and ways of structuring work.

If computers are to be involved in, and with, a wider range of interests than I have argued is the case today, fundamental changes are needed in that which sustains the current status quo of computing. Change is continuous and inevitable everywhere; the issue is notching it in certain directions. I remain optimistic that academics and practitioners trained in the social sciences are contributing to spreading awareness of the complexity (non-separation) of computing from e.g. work practice. Such challenges can go further in turning in on ourselves. (This may require some further explorations also of the situations within some social sciences than the previous ones have needed. A difficult but fertile area for co-work?) Am I asking others to do the "dirty" (hard, unpopular) work of challenging power? Maybe. But the issues seem too important to be left to computing professionals alone.

Postscript

I have conducted research that fits with the CSCW heading, and did for years feel at home in the CSCW community. But a comment from Joan Greenbaum crystallised my growing sense of unease: no matter what good intention researchers into CSCW have, those who buy these systems do it to save money. I expressed some of this unease in Beck 1996. This position paper is an attempt at digging deeper.

Reading the position papers of other participants brings it clear to me that a thumbnail sketch of the host of issues and debates assumed by my paper should perhaps be added. Many of those have been well presented in other papers. Christiane Floyd beautifully argues that software practitioners' self is integral to the work conducted and influences it; Markku Nurminen–as I read him–also has such issues at his chest.

A need for comp. & soc.sci'sts to work together has been articulated in some contexts. Marina Jirotka sets out some of the debate in CSCW, which has focussed on ways in which soc. and comp. scientists might collaborate. Jirotka's and Viktor Kaptelinin's papers are a reminder that the gulf I have assumed between comp.sci's & soc.sci's is a more complex issue than I have stated it above. And I am aware that my own relationship to the practice of software development is ambiguous and shaped by questions considerably more than may seem.

Yet, I retain a belief in the importance of my focussing on the political. I'm trying to pick up on what I see as one very basic condition for greater social responsibility in s/w practice (the desirability of which I will not argue right now): A culture in which seeing us as (jointly) responsible for probable (socio-political) effects of our work (Floyd's philosophical stances, the way I read her). I want to understand why this is evident for so few (or, if you like: "why have I come across so little evidence that s/w practitioners seem to agree with me?" :-).

Donna Haraway (1997) examines links between power interests and developments within biology (Genetic Engineering) and computing. She points out, among other things, chains of interest that link (some) universities with (some) multinational corporations. From a more individual(?) perspective, Phil Agre's (1997) introduction to an edited volume on people and computers sets out some of the conditions within which s/w practitioners work. This is the only published work I know of that starts to examine what may lie behind. His paper includes a discussion of parallels and contrasts to (other) engineering professions and points to issues like high pressures on keeping up-to-date in a rapidly changing field to get or keep a job.

Letting these influence a larger (and perhaps more cynical?) view of the interdisciplinary collaborations we are interested in here at the seminar, I can imagine further reasons why our discussion here is in vogue in the circles we are affiliated to: it may serve the interests of both sides, in ways that are more or less evident to those of us caught in it. (Some that I can imagine include the political conditions for soc.scientists in an increasingly pressured UK university sector in which moneys for research have been heavily cut over years, hitting social sciences heavily. This combines with a general political culture in Europe in which moneys for "pure" research have been increasingly hard to come by and research funders in many countries and in the EU centrally increasingly tie industry in as bedfellows of academia. Not all of this is necessarily bad, but the impact on what issues can and are brought up should not be underestimated nor silenced.)

Finally, in my paper I use a rhetoric which relies on a duality between "the" social sciences and "the" computing sciences and which is easy to pick apart. First, from a number of other perspectives than mine here, these two are not so different. Second, what are they anyway, that I could denote as entities? I pose a duality that does not exist except in a highly specific sense; I use and thereby reify these terms as icons of something I at the moment cannot express otherwise, leaving it to the reader to fill in a number of blanks. Third, I have problems placing myself in this landscape I paint. Still, my heart is lost in there somewhere between the desire to understand and the joy of constructing something "better", and like an artist trying to catch her dream in a painting I cannot let go of it until I have explored it further.

References:

Agre, Phil (1997) Computing as a Social Practice. In Agre and Schuler (eds) Re-inventing Technology, Rediscovering Community. Critical explorations of computing as a social practice, pp.1-.

Beck, Eevi (1994) Practices of Collaboration in Writing and Their Support (D.Phil Thesis, U. of Sussex, tech.report CSRP 340, ISSN 1350-3162, School of Cognitive and Computing Sciences).

Beck, Eevi (1996) P for Political? Some Challenges to PD towards 2000, in Proceedings of PDC'96.

Beck, Eevi (1997) Managing Diffracted Rationalities: IT in a Home Assistance Service, in Moser & Aas (eds) Conference on Technology and Democracy: Gender, Technology, and Politics in Transition? Tech.report 29/1997, TMV skriftserie, Centre for Technology and Culture, Univ. of Oslo, pp.109-.

Haraway, D. (1997) Modest_Witness@Second_Millenium.FemaleMan Meets_OncoMouse. Feminism and Technoscience (New York and London: Routledge).

Orlikowski, Wanda (1992) Learning from Notes: Organizational Issues in Groupware Implementation, in Proc. CSCW'92, pp.362-.

Sørensen, Bjørg Aase (1982) Ansvarsrasjonalitet: Om mål-middeltenkning blant kvinner [loosely from Norwegian: Rationality of responsibilities: On means-end thinking among women], in H. Holter (ed) Kvinner i fellesskap, pp.392-402 (Oslo: Universitetsforlaget).

Mediation and Heterogeneity in Design

Olav W. Bertelsen

If I see you it is because I want to hit you.
—Marx Wartofsky

In the social practice of software thinking, formalism and formalisation are understood both to be core aspects of computing and problematic themes when people are taken into account. Large parts of research in the area of systems development, co-operative design, human-computer interaction and CSCW have been concerned with the impossibility of formalisation of work. Social studies of work in an office setting (Suchmann & Wynn 1984, Suchmann 1986) have been the cornerstone in the critique of Tayloristic office automation. The UTOPIA project (Bødker et al. 1987) showed that formal descriptions did not work in the co-operation with users from the graphical industry. Consequently, techniques such as organisational games, mock-ups, and rapid prototyping were developed (Greenbaum & Kyng 1991) in order to overcome the difficulty through concrete enactment of possible futures with the proposed technology. The criticism of the "waterfall model" is included in most textbooks today, indicating that even in organising the work of computer professionals, formalisation has limitations.

A specific criticism of formalisation in software development is that of Peter Naur (1985), who argues that, computer programs only make sense to the team who have developed it. According to Naur, a theory about what the program does, and how it does it, is built simultaneously with the construction of the executable code. This theory cannot be written down or otherwise formalised, and is only accessible to the programmers working on the particular project. Programming has the double character of being both construction and conception. Consequently, software cannot be specified and described formally, but depend on personal insights. While Naur makes a strong point about the power of formal specification, he neglects notions of social mediation and heterogeneity in general. To understand and support software design we need to include notions of social mediation and heterogeneity. Furthermore, since we have to deal with formalisation in a non-formaliseable world, we need to address the limitations of formalisation.

Design is heteropraxial, i.e., involving heterogeneous groups of people with different backgrounds and different motivation for participating in the process. Co-operative prototyping (Bødker & Grønbæk 1996) can serve as a general metaphor for design. In idealised terms, prototyping is a process where one or more designers work together with one or more prospective users on developing new computer support for the users. Based on some sort of vision or an analysis of the users' needs the system developers build the first prototype. Users and designers work with the prototype in work (like) situations, and they gradually adjust it to fit the (renewed) praxis. They are fundamentally unable to understand each other, but during the prototyping session, they build a common understanding that is only present as the final prototype. In an activity theory terminology (Engeström 1987), design can be understood as an instrument producing activity in relation to a considered central, use activity, and the use activity can be understood as object activity for the design activity. This analysis, however, does not directly address the intertwined character of the prototyping session proposed as a general metaphor for design. What tie the involved activities together in design are the involved artefacts; the design artefacts and the artefacts that are object of the design process. In the design situation, these artefacts become boundary objects, constituting a boundary zone of design, where users and designers meet to change the world together but not necessarily understand each other.

Activity theory states that human conduct is mediated by artefacts (Engeström 1987). Accordingly, design is mediated by design artefacts: programming environments, design methods, specific techniques, theories, etc. These design artefacts are boundary objects. They tie involved activity systems together, and they tie different rooms of design and use together. Boundary objects (Star 1986) are at the same time robust enough to maintain identity across use by different parties, but also plastic so that they can adapt to the different parties needs and constraints. Boundary objects mediate the relation between divergent viewpoints; being weakly structured across site, but highly structured in specific context. The heterogeneous activity systems contributing to design are tied together through their joint use of artefacts and through their joint focus on the same object. Different motives drive users and designers; the object of design does not make sense in the same way for them. Design artefacts are boundary objects in the sense that they tie different praxes together, maintaining meaning across groups but making sense in different ways. When designers and users work together on a system specification, designers may perceive the specification as a data flow model, whereas users may understand the same specification in terms of new ways of working in the organisation. In the same way, the double character of design artefacts described above also makes them boundary objects within praxis. Design artefacts do not only take different shapes or serve different purposes in different groups, they also take different functions within one group across time, during use and design, and in the different rooms of a specific group's praxis. Thus, a system development method is a boundary object in the sense that it has one function in the project organisation's internal education prior to a project, and another function during the project. Working in accordance with the method means two different things in the two project rooms. In the same way as design artefacts are boundary objects, the object of design is a boundary object. Realising that design artefacts and the object of design are boundary objects, makes it possible to revise common ideas of shared understanding and interpretation in design. The crucial point is not for the designers to be able to understand and interpret the users correctly, but to supply design artefacts that can serve as boundary objects in mediating the design process.

Design artefacts have a double character of both mediating the technical construction of computer systems, and mediating representational activity, on one hand formalised, on the other depending on openness for interpretation. This double character is a problem because the plasticity of the representational side is obstructing or obstructed by the naturalism and formalisation of the technical construction. The example below illustrates this feature of design artefacts.>From a technical perspective, a design method like Structured Analysis and Design (Yourdon 1982) is as a very high-level program construction tool. However, at the same time this tool serves as mediator of communication and conception. The applicability of elements from structured design in communication and conception with users depends on the designers' success in establishing common representations based on the tool. Based on qualitative interviews Laursen et al. (1990) report how output from a CASE tool both facilitate and obstruct user designer co-operation according to various circumstances. In one particular situation, a context diagram(poetically referred to as the "the sunflower") was successfully applied as a frame of reference shared by the whole project group during the project. In another situations the same designers had no luck trying to apply diagrams from the same CASE tool in communication with the same users. In both situations, the CASE tool and its output had a double role, both mediating production of machine executable code and mediating communication and conception. In the first situation (the sunflower), the formalised features were weak enough to allow the formalised description to acquire other meanings. In the other situation, however, the model was so complicated that it was impossible to deal with beyond merely understanding the formal contents of the figure. The "sunflower" offered openings into a poetic world, whereas other diagrams only generate frustration for the users. The "sunflower" was a boundary object in the sense that designers perceived it as a computer system description whereas users perceived it as an image of the organisation; still, it maintained stability across these different perceptions.

We may expand the linguistic distinction between *meaning and sense* to the analysis of tools. The meaning of a hammer is the assemblage of shaped iron and wood. The sense of the hammer is that carpenters use it for driving nails. The sense of the hammer is "crystallised" in secondary artefacts (Wartofsky 1979), e.g. anecdotes and sayings, mediating hammering praxis. The meaning of the "Sunflower" described above was that of a (formalised) SA/SD context diagram. The sense of the "Sunflower" in the collaboration between designers and users was the context of the prospective computer system, and the way it would mediate parts of the relations between activities in the organisation. Naur (opcit) based his programming as theory building idea on exactly the same doubleness. It is a secondary artefact (Wartofsky op cit) conserving the acquired knowledge and skills in working with the program. However, Naur bases his analysis on the individualist philosophy of Ryle, neglecting the societal/cultural nature of representation and mediation in general. The point in the concept of secondary artefacts is that human modes of relating to the environment (including programs) in culturally developed and mediated. Secondary artefacts not only conserve knowledge and skill among the individuals whose experience they are based on, but secondary artefacts also *transfer* these across a given culture. In making the argument that, the "theory" of a program cannot be passed over to another team; Naur (op cit) refers to a number of empirical cases. These cases, however, are ambiguous, and could as well be an argument for the deficit of established design artefacts. Wartofsky's (op cit) argues that it is a basic feature of human praxis that experiences can be shared and transferred, his argument cannot be rejected based on Naur's program theory idea.

In looking at the design process as such, heterogeneity and mediation are reflected in the concept of "design as transformation of artefacts" (Bertelsen 1996, 1998). It is a process-oriented concept, focusing on the connection between development in use and development by design. In general, activity with one generation of an artefact is crystallised into the next generation (Bærentsen 1989); thus the history of a concrete praxis can be discovered through the history of the mediating artefacts. The idea of transformation is to base design on the historical development of praxis by deliberately continuing the crystallisation of activity into artefacts, concretely bringing "object history" (Engeström 1987) into design. Use and design are seen as two sides of the same coin by turning artefacts mediating use praxis into local design artefacts, and back again. The process of taking central artefacts from the considered praxis into the zone of design depends on the boundary object capability of these artefacts, their ability to maintain identity across use and design. When the central artefacts have been brought (successfully) into design, they are gradually changed while they are mediating simulated praxis. By confronting the artefacts with praxis throughout the process, it is ensured that the subsequent transformed versions make sense in praxis. The process of bringing central artefacts from praxis into design is a representation process, depending on the stability of boundary objectness across representation. During transformation artefacts exist both in their original form within the domain of use, and (often as representations) in the domain of design. The artefacts being transformed are boundary objects in the sense that they exist in different activity systems, and tie these together across groups, phases of praxis, and use-design. The concept of transformation is a descriptive one based on the analysis of successful as well as failed co-operative design projects. It illustrates how use and design share similar features of development, mediatedness and heterogeneity.

Concepts of heterogeneity are gaining increased importance in the CSCW community. Concepts like boundary objects (Star op cit.), and double level language (Robinson 1989) are widely applied and agreed on in the field. Likewise, the concept of common (or shared) information spaces (e.g. Bannon & Schmidt1991) is used to emphasise a focus on the negotiability and context specificity of information and the heterogeneity of information repositories. Both concepts imply a bias towards representation and interpretation, away from material conditions and formal constructs. In the system theoretical era, e.g. traditional office automation, representation in software design was a matter of digging out the one correct picture (Bansler 1989). The newer CSCW concepts serve as important instruments in developing alternatives to such formalist foci. It is; however, important to acknowledge that software is implemented in technical structures, often hardwiring specific interpretations as the one only fact. Between material fact and subjectivist interpretation, between the physical and the symbolic, the concept of design as transformation of artefacts, described above, is one account on how these aspects are intertwined in use and design. Wartofsky (op cit) argues that perception is a mode of outward action, and that representation is part of the cultural maintenance and development of productive praxis. Because representation is inseparable from action, we need to acknowledge modes of representation that are not linguistic or symbolic. This may be the key to solving the problem of software documentation and design support in general, including communication of Naur's program theory.

In the search for an integrated perspective on design and use, beyond hard systems thinking, the concepts of heterogeneity and mediation supplement each other in the space between naive engineering materialism and subjectivist post-modern social research.

References

Bansler, J. (1989). Systems Development Research in Scandinavia: Three theoretical schools, *Scandinavian Journal of Information Systems* vol.1, Aug. 1989, pp. 3-20.

Bannon, Liam, and Kjeld Schmidt: 'CSCW: Four Characters in Search of a Context,' in ECSCW '89. *Proceedings of the First European Conference on Computer Supported Cooperative Work, Gatwick, London, 13-15 September, 1989*, pp. 358-372. - Reprinted in J. M. Bowers and S. D. Benford (eds.): Studies in Computer Supported Cooperative Work. Theory, Practice and Design, North-Holland, Amsterdam, 1991, pp. 3-16

Bertelsen, O. W. (1998). *Elements to a theory of design artefacts: a contribution to critical systems development research*, Ph.D. Thesis, Aarhus University. DAIMI PB531. (available from http://www.daimi.au.dk/~olavb)

Bertelsen, O. W. (1996). The Festival Checklist: design as the transformation of artefacts. In Blomberg, J., Kensing, F. & Dykstra-Erickson (eds.). *PDC '96, Proceedings of the Participatory Design Conferenc*e. Palo Alto: Computer Professionals for Social Responsibility. pp.93-101.

Bærentsen, K. (1989). Menneske og maskine [Man and Machine]. In Hedegaard, Hansen & Thyssen (eds.). *Et virksomt liv* [An active life].Aarhus: Aarhus University Press. pp. 142-187.

Bødker, S., P. Ehn, J. Kammersgaard, M. Kyng, & Y. Sundblad (1987). A UTOPIAN Experience: On Design of Powerful Computer-Based Tools for Skilled Graphic Workers. In Bjerknes, G., P. Ehn, & M. Kyng (eds.). *Computers and Democracy*. Aldershot UK: Avebury. pp. 251-279.

Bødker, S. & Grønbæk, K. (1996). Users and designers in mutual activity: An analysis of cooperative activities in system design. In Engeström, Y. & Middleton, D. (eds.). Cognition and Communication at Work. Cambridge: Cambridge University Press. pp. 130-158.

Bødker, S. (1998). Understanding representation in design. *Human-Computer Interaction* 13 (2), pp. 107-125.

Engeström, Y. (1987). *Learning by expanding: an activity-theoretical approach to developmental research*. Helsinki: Orienta-Konsultit Oy.

Greenbaum, J. & Kyng, M. (eds.). (1991). *Design at Work*. Hillsdale, N.J.:

Laursen, B, et al. (1990). *Dokumentation & Kommunikation ioliebudgetprojektet*, [Documentation and Communication in the oil-budgetingproject.]. unpublished case study report, Computer Science Department at Aarhus University.

Naur, P. (1985). Programming as theory building. In *Microprocessing and Microprogramming* vol. 15, pp. 253-261.

Robinson, M. (1989).Double Level Languages and Co-operative Working, *Conference on Support, Society and Culture. Mutual uses of Cybernetics and Science - Proceedings. Amsterdam 1989.* pp. 79-114.

Star, S. L. (1989).The Structure of Ill-Structured Solutions: Boundary Objects and Heterogeneous Distributed Problem Solving. In Gasser, Les & Michael N. Huhns *Distributed Artificial intelligence*, volume II. London: Pitman Publishers, pp. 37-54.

Suchman, L. and E. Wynn(1984) Procedures and Problems in the Office. In *Office: Technology and People,* vol. 2, 1984, pp. 133-154.

Suchman, L. (1986). *Plans and Situated Actions.* Cambridge MA:Cambridge University Press.

Wartofsky, M. W.(1973). Perception, representation, and the forms of action: toward an historical epistemology. In Wartofsky, M. W., *Models*. Dordrecht: D. Reidel Publishing Company, 1979. pp. 188-210.

Yourdon, E.(1982). *Managing the system life cycle*. New York: Yourdon Press.

Design and Social discourse

Thomas Binder

The title of our seminar indicates two interesting and partly opposite currents in the on-going debate on design of information technology. 'Social thinking' lets us think about the growing bulk of interest and experience gained within our field through encounters with problems and professional approaches traditionally dealt with in the social sciences. 'Software practice' on the other hand hints at the fact that we since the advent of the field of CSCW has become painfully aware of the complexity of professional work and not least the work involved in producing software. I have tried to capture a take on these themes with my title: Design and Social discourse. 'Design' is there to indicate that for me the central professional issue is (still) the skilful joining together of possible 'bricks' for an 'imagined future' (as a corrective or commentary to the call for 'engineering'). 'Social discourse' on the other hand should give a slight twist to the aspirations one could have in terms of getting on top of the social issues flowing into our field. I will develop my position in three steps. First I will argue that design is still the link we need to connect thinking with practice. Second I will try to come to grips with how social discourse direct the way we are able to engage in design. And third will I make a hazardous attempt to address some of the design challenges we are facing.

Design is (still) the missing link

Herbert Simon was among the first to raise the issue of design as a response to the mismatch between scientific knowledge and the skilful practice of (university trained) practitioners. In his book 'Science of the Artificial' he suggested a new professional ideal for these practitioners. On one side they should be trained in a rigor of thought learned from the sciences, but on the other side they should be equally aware of that crafting artifacts can only be evaluated for its purposefulness. The task of the practitioner is according to Simon the job of a designer of interfaces. Purposefulness does not stem from a particular application of scientific knowledge but from the careful bringing together of various sub-systems whatever of social or scientific origin. Problem-solving becomes the key issue, and the direction for the science of the artificial is accordingly pointed out as the rigorous exploration of procedures and strategies by which the practitioners can find 'good enough'-solutions to 'ill-structured' problems.

Behind Simons reasoning one senses the image of engineers or bureaucrats in large organisations where the goals are clear and professional competency has to be invested in finding ways to achieve them. This image seems well in line with the post-war thrive towards uniform growth along a few well elaborated paths (the welfare state or the mission to the moon), but leaves other practitioners such as doctors, architects or musicians in the dark. Donald Schön addresses some of the problems that Simon left out. He shares with Simon the idea that designing is the core skill of the professional, but where Simon tends to think of designing merely as the process of joining together solutions, Schön includes the setting of the problem as a crucial part of the professional design process. For Schön the exemplary designer is the architect who has to create an image of a yet unbuild building in confrontation with site and client. The practitioner does not know the problem from the outset, and she can not derive it in any strict sense from neither the clients task description nor the properties of the site. The example is parallel to Simons famous interface issue in the design of a clock. A purpose (pleasurable living/ knowing the time) has to be interfaced to a natural system (the site/the inner workings of the clock). Simon however tends to deal with the design problem as a question of 'taking in' information where Schön suggests a more outward leap. For him what counts is how the designer must impose herself on the problem situation in order to get the setting to 'talk back'.

Schön's reflective practitioner has become an attractive model for the kind of design thinking that is needed for the new professions struggling with an unknown problem field. Design of large information systems seems to call for precisely the kind of reflections on problem setting that Schön suggests. Within the professions engaged in development of information systems a 'Schönian' modeling of professionalism opens up for an attractive self-image as information system architect. In some strange way this image brings together promises of new and fruitful approaches to wicked problems (problem settings as described in specs are no longer right) with old aspirations of control (we can make the system as right as a Le Corbusier building). Schön did not push his point in that way. His focus on the individual design process facing unique design problems does however make his writings less illuminating for design activities involving many participants.

Why then is design (still) the missing link? If the problem we are struggling with is rightly interpreted as: How do we base and inform software practice in a (in the best sense of the word) scholarly rigor of thought? then we are basically still wrestling with Simons problem of the nature of the artificial and with Schöns emphasize on problem construction. Rather than throwing away the label of design for its director connotations, I think we can gain from staying with design as the term used to encompass the inevitable leap from insight to construct or action inescapable in any practical profession.

Shifting discursive terrain: From 'users' and 'tools' to 'environments' and 'props'

In the Scandinavian tradition of participatory design within which I have my roots, we have come quit far in pointing out that Simons 'troopers' missed out not only important players in the defining game of purposefulness but also an elementary sense of context. It has convincingly been demonstrated how conventional design projects fail when equalling design intentions with resulting outcome, and we have come to a point where we can say that exploring technology in use and involving users in design is actually something we can do. The icebreaker in this process has by and large been a utilitarian insistence on questioning the use-qualities of both ones own and the opponent's artifacts. This rhetoric is not in any way foreign to the rationalistic discourse, which also Simons writings are a part of. We can discuss to what extend we have arrived at a point where 'true' user-involvement in design projects has become acceptable in industry, but to me there is no doubt that the notions of 'use' and 'users' has become part of the software engineers standard vocabulary. We have not reached the 'platonian republic' of design that Schön spotted as the underlying utopian ideal of the participatory design tradition (Schön in Binder, 1996 (1)), but we have surely been part of a process that let most engineers today speak freely of the politics of design. A strange outcome of this process is however that 'use' and 'users' has become much more doubtful rhetoric allies in the strive for an open-minded and innovative approach to design of information technology (Binder 1996).

From my own work on learning environments in factory settings I have found that the focus on use and users becomes an almost untrancendable stepping stone for the 'work system' already in place. Thinking of the use of the artifacts in utilitarian terms inevitably leads to the need for matching up with a more or less fixed system of task, and offering user-involvement narrowly defines the roles of those to whom the new environments should be a place of discovery and exploration. Getting beyond this stalemate situation calls for redefining what our design projects are about (Binder 1997). In the emerging field of ubiquitous computing where I am at present trying to gain ground for a design and research practice informed by the PD experience we are in constant discussion on the relevance of engaging with people in the settings for which we do our designs. Some of my younger colleagues seem to see the field in it self as the 'freeing of the machine' which make them able to work innovatively and (I would say) almost futuristically with all new types of settings where HCI conventions such as 'screen and keyboard' are left behind. In my reading much of what we have seen up till now of ubiquity has a clear imprint of the 'life-world' of the designers involved, and in my view we could get a lot further if we find ways to base our designs on environments which are foreign to us. On the other hand I can see that the plurality of interfaces (and devices) in themselves seem to deny a high profiled user-centeredness simply because use of the artifact takes on a new and far more modest role in the target setting. Most lately we have been working with distributed computing in industrial process plants, and here the idea of fixed tasks and users simply does not give relevant clues for design thinking (Binder 1999).

The move we have to make as designers and design researchers is to de-center the focus of our work. We shall not be designing for particular users but for a set of artifactual environments and we shall not be designing tools capable of handling particular tasks but rather 'props' which people in the setting can invest in order to invoke relevant practices. As I see it such a move is well in line with the way society as a whole is moving. We are moving away from both a monolithic tale of technological progress (in which Simons writings are easily inscribed) and an artistic perspective on design emphasising the expressiveness of the designer (as Schön's designer might be interpreted).

Design between community storytelling and prospective archaeology

To change attention from 'users' to 'environments' does neither mean to disengage with 'context', or to do away with the dialogue with the people in it. Rather the shift implies a need for emphatic engagement and exploration at the same time giving up the attempt to 'fix a problem' and acknowledging that this engagement can only be successful if we as designers impose ourselves on the setting. Heuristically one could say that we shall learn to 'tell the environment' by employing and expanding the metaphors that are available to us.

Similarly the more modest claim of doing 'props' rather than 'tools' is not taking anything away from the mental investment we have to do in our artifacts. We have to learn an almost archaeological capability of reading opportunities into artifacts not yet designed. In doing this we must also give up any idea of our intentions (or even readings) having the power to manifest themselves in a practice we can foresee (Reddy has a nice discussion of the playground for these considerations in his discussion of the conduit metaphor. Reddy, 1985)

Finally and of no less importance we have to come to terms with the new and changing circumstances where we have an intricate mirroring of artifact and design process in a setting where designers are not in line. Where the terminal- work place can be seen as the perfect image of the post war technological regime, around which a large number of disciplinary concerns could be exercised, we are now operating in an open field which shows little signs of stabilising around new strong images. Therefore we have to re-elaborate on Simon's suggestions for a science of the artificial, doing away with its underlying currents of bureaucratic rationality, but working as hard as ever to understand how artifacts facilitate human action. We also have to revisit the notion of the designer as reflective practitioner. This must be done with the insight that designing is an all pervading activity of many communities in which we have to construct not only the problem and the artifact but also the 'rules of the game' by which we do our 'moves'.

Literature

Simon, Herbert A., The Sciences of the Artificial, MIT Press, 1976

Schön, Donald A., Educating the reflective Practitioner, Jossey-Bass, 1987

Binder, T. , Learning and Knowing with Artifacts, an Interview with Donald Schön, AI & Soc (1996), 10:51-58, Springer 1996

Binder, T Designers in Dialogue, in Vinck and Perrin (eds.), The Role of Design in the Social Shaping of Technology, COST A4, EU , 1996

Binder, T. Designing for Workplace Learning, AI & Soc (1995) 9:218-243, Springer , 1997

Binder, T., Setting the Stage for improvised Video Scenarios, Proceedings of the CHI 1999 conference, 1999

Reddy, Michael, The Conduit Metaphor, in Metaphors and Thought, Cambridge University Press, 1985

Changing work practices in design

Keld Bødker

Together with Finn Kensing and Jesper Simonsen, I have developed a method for design in an organisational context (Kensing et al., 1998a). The method has been developed based on a combination of theoretical and empirical studies. In the empirical studies we - as designing researchers - have carried out design projects in various organisations in Denmark and the US in cooperation with designers and users from the involved companies (Bødker and Kensing, 1994; Kensing et al. 1998b; Simonsen, 1997; Simonsen and Kensing, 1997).

To test the method we recently carried out projects in three Danish IT-organisations. Our role in these projects was restricted to method dissemination – analysing the current work practices, propose changes, teaching, supervision, and coaching – and to reflect on the experiences in relation to changing work practices of designers doing real life industrial projects.

The MUST method deals with design of IT in an organisational context. Design in this context, as systems development and implementation in general, involves changing work practices in the user organisation. So designers are accustomed to change. However, with the present projects we deal with a second order change. Design – as system development and implementation – contributes to changing an organisation's present work practices into a new form. Correspondingly, changing the work practices of designers means changing present forms (present design work practices) into new ones (new design work practices). It is often claimed that it is harder for designers to change their own work practice than to change the work practices of others, i.e. the work practices of the users of the designed systems.

One of the premises of the projects are that teaching of new design techniques cannot stand alone, but has to be supplemented by supervision and coaching. To many practitioners, the new work practices of detailed observation and close cooperation with various professionals in their role as future users are so new that they have to be supervised and coached in the same way as students doing project work.

Also, it is our experience that the decision to use a new method or new techniques should be coupled very tightly to experienced problems with the present ways of working. Finally, our experiences indicate that real change is most likely to happen in relation to direct contact with an ongoing project where the new design practices can be applied and tested.

References

Bødker, K. and Kensing, F. (1994). Design in an Organizational Context - an Experiment. *Scandinavian Journal of Information Systems,* Vol. 6, no. 1, April. 47-68.

Kensing, F., J. Simonsen and K. Bødker (1998). MUST: A Method for Participatory Design. *Human-Computer Interaction,* 13(2), 167-198. (1998a)

Kensing F., Simonsen, J., and Bødker, K. (1998). Participatory Design at a Radio Station. *CSCW: The Journal of Collaborative Computing,* Vol. 7, no. 3-4. (1998b)

Simonsen, J. (1997). Linking Design to Business Strategy through Functional Analysis. In R. Galliers et al. (Eds), *Proceedings of the 5th European Conference on Information Systems,* 1314-1327. Cork, Ireland: Cork Publishing Limited.

Simonsen, J. and Kensing, F. (1997). Using Ethnography in Contextual Design. *Communication of the ACM,* Vol. 40, no. 7, July, 82-88.

Work Practice and Participatory Design of Software Development

Yvonne Dittrich

Two years ago I moved from the University of Hamburg to a small university in southern Sweden. Having a background in participatory and evolutionary software development, I had high expectations. Working in one of the countries where the 'Scandinavian Approach' has developed, I was looking forward to discussions both with the work scientists of our group and with my computer science colleagues. After a short while, I discovered that the gap between the representatives of the 'Scandinavian Approach' and the 'normal' computer scientists was even deeper than in Germany. Besides being split up into different departments at many universities, I perceived mutual prejudices, separating the, for me, so closely related issues of participatory design and development of software. In industrial practice here in Sweden I perceive more openness for issues like use oriented design and participatory design than in Germany. However, in the software development practice the language of ISO 9000 and the 'Capability Maturity Model' (CMM) is quite dominating.

Despite the differences in language, I find myself relatively close to practitioners, when it comes to concrete issues in software development. A colleague of mine working part time for industry took part in a project implementing an 'experience engine', a small 'experience factory' [2]. Half way into the project they cancelled the development of a measurement database and implemented an organisational role caring for developers with specific experiences and those who want to learn about the same subject to meet. [20] In another case, a department of a nearby software development company used a qualitative study of a 'method and tool' development project to reflect on and try to improve their own practice. I begin to read the, for me strange literature of the CMM and Software Engineering Laboratory (SEL) with different eyes, and a different interpretation. In the following my ambition is to formulate a stepping stone between the use-oriented design and development of software, empirical research in software engineering, and software development as work. And I will try to relate it to a change in perspective that I see lurking at the horizon.

Software engineering as a research area aims at developing methods, tools and other means to enable software developers to do a good job, to develop high quality products in a not too stressful way. For a majority of software developers the term 'quality' refers to product quality as well as quality in use. Most software developers aim at developing useful programs – even as the results are often frustrating, both for users and developers. On the other hand, especially regarding user centred design and participatory design, a large gap between research communities and industrial practice is observable. Workshop themes like 'How to make user centred design usable' at the INTERACT '99 in Edinburgh indicate a need for relating our research results to industrial practice. Although there are many accounts of the conditions for participation within the use organisation, few considerations take the developmental side into account. However, participatory design and development does not work without the developers.

Software engineering: a scientific design of a complex work practice?!

Compared to other disciplines, software engineering has a quite specific history. It did not develop out of an established practice asking for abstractions, concepts, and models to support further developments. It was founded because of the lack of such a good practice. Software engineering started out as a scientific design of a complex work practice. The different perspectives emphasising different aspects and developing in different scientific discourses were visible from the very beginning:

Choosing the term software **engineering** for a new research area 1967 was meant as a program. 'The whole trouble comes from the fact that there is so much tinkering with software. It is not made in a clean fabrication process, which it should be. […] What we need is software engineering.' [4] As an answer to the software crisis, the development of software should be improved, using the engineering disciplines – especially electrical engineering – as a paradigm. [16] Methods and tools should be developed providing reliable products as results of a controlled and predictable process. Research emphasis was put on formal notations and a structured programming and the division of programs into modules allowed for a division of labour as well. The development is perceived as the engineering of a single product, requiring creativity and experience of the engineers. The goal was to provide suitable concepts and methods to support them.

Osterweil's article 'Software Processes are Software Too' [24] can be regarded as a concise presentation of another paradigm for software development. In his article software is described not longer as a product of a group of creative and experienced professionals, but as the outcome of a quasi-algorithmic process. The software development as **industrial production** was discussed already at the conference in Garmisch-Patenkirchen 1967. [22] Formal and as complete as possible early specifications are stepwise transformed into the final program. Computer professionals working on an intellectual assembly line implemented by the software factory, a case tool integrating planning and co-ordinating the distributed work. [11] The outcome is controlled by quality assurance measures borrowed from industrial production. Measurements are not only input for research, they are meant to be used in practice to control and tune the development process. [2]

The **design** perspective was represented at the Garmisch-Patenkirchen conference as well. [4] At first the concept was only related to the constructive side of software influencing the development of the field of software architecture. Using design in a wider sense, it became important regarding user centred design. Quality in use became criteria for software development. Developing a useful and aesthetic technical artefact is perceived as finding a match between technical possibilities and problems posed by the use context. [12, 13, 14] Again architectural design – for example [1, 30] – is used as a prototype to understand this match making. Methods and tools are reflected as framing the perception of the problem as well as the solution space [21]. Evolutionary process models are meant to support a co-development of work practice and information technology. Empirical base is own experience and industrial projects where together with practitioners new methods are developed and tried out.

Software engineering research has been successful in supporting and informing the practice with methods and concepts. Even as formal methods, software architecture, project management, and user centred design are still subject of conferences, they are so on a different level. Nonetheless, the tension between research and practice is not yet resolved.

Software Development Practice: informed by but not mimicking the scientific paradigms

In practice the different perspectives are often merged. Methods are used in a situated way: The engineering paradigm still seems to be the main one in the self-recognition of software developers. Measurements and quality assurance systems based on the industrial production paradigm are widely applied. The design perspective is emphasised regarding software architecture – the design pattern movement changed the practice of object oriented programming in a revolutionary way – as well as regarding the user interface and the embedding into work practice. On the other hand, software-developing companies earn money despite not using formal methods [25], working on CMM level 1 [26] and ignoring participatory design.

Some of the differences between academia and practice might be explained by a normal time lapse between research and industrial application. The observable merge between the different paradigms, however, points to the development of an independent practice. This practice poses new questions. Why do some things work out, and others do not? Why are methods applicable in one context but do not work in another? Why does participatory design work in university projects but is considered a high risk in industrial practice?

Software Engineering goes Empirical

Also mainstream software engineering research is beginning to realise the problems with the above approach. As a contribution to the 25th anniversary of the discipline IEEE Software published beside other articles a text 'The Software Research Crisis' where the author – fictively evaluating today's situation from 25 years later – claims that the software research now (2020) has overcome its crisis. He attributes most of the existing software engineering research as arrogant and narrow: researchers who never worked in practice claiming to know what is good for that practice, advocating their mindchilds without taking the use context of them seriously. [15] Glass is cites a number of articles discussing the research paradigm of software engineering, arguing for a more empirical, more practice-oriented software engineering research.

He then puts forward the SEL approach [3]: an innovation is applied experimentally in a project, analysed and evaluated regarding the experiment's goals. The gained insight is used to improve the innovation, apply it again, evaluate it, and so on. That the analysis and evaluation of the methodological innovation uses quantitative measurements seems to be taken for granted and is not even discussed.

Software development, however, is a social activity. Most projects are unique in their setting and the proposed outcome. The application of quantitative methods only restricts the possible outcome to the narrow area of what is measurable. The difficulties and criticism of a quantitative paradigm within social sciences applies for the empirical research on software development as well.

Co-operative design as a theme in CSCW

As software development is a co-operative effort, software development has become subject of the CSCW discussion. [29] Traditionally empirical research in CSCW mainly uses qualitative methods: Ethnography, ethnomethodologically informed ethnography, for example, often combined with participatory design processes as action research.

Articles raise issues like the use of representations and design work as embodied [31, 6], organisational constraints and their influence on the work practice [5, 7, 8], the development of organisational patterns from within the project group [27, 28, 32], or about the interaction between work practice and computer based tools [17, 18]. It is perhaps too early to expect a feed back into software engineering literature. However, the relation between research on co-operative design and the empirical approaches in software engineering look a great deal like mutual ignorance.

Methods as tools, and how to design them in a participatory way

Beside user centred design and development of software, especially the 'Scandinavian Approach' might contribute as an approach to software engineering research itself. If methods are understood not as a prescription of action, but as a resource for the software developers, the same arguments apply for the development of methods as for the development of software.

The computer based, organisational and semiotic tools for software development might become more usable if they are design, implemented and evaluated in a participatory way. Methods can be understood as providing support for a certain way of doing software development and not lending itself easily to others. On the other hand, as detailed and prescriptive as a method might be, it has to be interpreted and adapted by the software developers regarding the circumstances and contingencies of the concrete situation. The actual implementation has to be designed in use. [10] Learning from participatory design and evolutionary development, method development can be organised as design – implementation – evaluation cycles, providing space for learning among all participants, practitioners as well as researchers.

Qualitative research about software development as work can as base for the change efforts and as means for evaluation open up for empirical results beyond quantitative measurements. Aiming at an intrinsic understanding of a specific practice ethnographic, and ethnomethodological approaches can be used to overcome prejudices of how software is, or should be, developed. Projects like the one described in [19] are exploring this kind of research. Lets follow them in founding software engineering in empirical research and in taking our users serious.

References

6. Alexander, C. *The Timeless Way of Building*. New York: Oxford University Press, 1979.
7. Basili, V., Caldiera G., and Rombach, D. H., The Experience Factory *Encyclopedia of Software Engineering,* 2 Volume Set, pp 469-476, John Wiley & Sons, Inc., 1994.
8. Basili, V. Greeen, S. Software Process Evolution at the SEL. *IEEE Software* July 1994: 58-66.
9. Bauer, F.L. Software Engineering – wie es begann *Informatik Spektrum* (1993) 16:259-260.
10. Button, G., Sharrock, W. Occasioned practice in the work of software engineers. In Jirotka, M., and Goguen, J. *Requirements Engineering. Social and Technical Issues.* London 1994
11. Button, G., Sharrock, W. The Mundane Work of Writing and Reading Computer Programs. In: ten Have, P., Psathas, G. (eds.) *Situated order: Studies in the social organization of talk and embodied acticities*. Washington DC, University Press of Amerika 1994 (5?)
12. Button, G., Sharrock, W. Project Work: The Organisation of Collaborative Design and Development in Software Engineering *Computer Supported Cooperative Work, Special Issue on Studies of Cooperative Design*. Vol. 5, No.4, 1996: 368-386.

13. Button, G., Sharrock, W. The Organizational Accountability of Technological Work. *Social Studies of Science* 28/1 (February 1998), 73-102.
14. Carstensen, P. H., Sørensen, C. From the Social to the Systematic. Mechanisms Supporting Coordination in Design *Computer Supported Cooperative Work, Special Issue on Studies of Cooperative Design*. Vol. 5, No.4, 1996:387-413.
15. Dittrich, Y. *How to make sense of software – interpretability as an issue in design.* Technical Report, University of Karlskrona Ronneby 1998.
16. Evans, M. *The Software Factory – A Fourth Generation Software Engineering Environment.* USA: John Wiley & Sons 1989
17. Floyd, C. Outline of a Paradigm Change in Software Engineering. In: Bjerknes, G., Ehn, P., Kyng, M. (Eds.) *Computers and Democracy*. Aldershot 1987, pp192-210.
18. Floyd, C. Software development as reality construction. In: Floyd, C. et al. (eds.) *Software Development and Reality Construction.* Springer Verlag: Berlin 1992.
19. Floyd, C. Software-Engineering –und dann? *Informatik Spektrum* (1994) 17:29-37.
20. Glass, R. L. The software Research Crisis *IEEE Software* November 1994: 42-47.
21. Goos, G. Programmiertechnik zwischen Wissenschaft und industrieller Praxis *Informatik Spektrum* (1994) 17:11-20.
22. Grinter, R. E. Supporting Articulation Work using Software Configuration Management Systems *Computer Supported Cooperative Work, Special Issue on Studies of Cooperative Design*. Vol. 5, No.4, 1996: 447-465.
23. Grinter, R. E. Recomposition: Putting It All Back Together Again *Proceedings of the Computer Supported Cooperative Work CSCW 1998,* (Seattle, November 1998) ACM Press.
24. Iversen, J. K., Nielsen, P. A., Nœrbjerg, J. Problem Diagnosis Software Process Improvement. In: T. J. Larsen, L. Levine, J.I. DeGross (eds.) *Information systems: Current Issues and Future Changes* IFIP 1998.
25. Johansson, C., Hall, P., Coquard, M. Talk to Paula and Peter – They are Experienced *Proceedings of the Workshop on Learning Organizations* June 16, 1999, Kaiserslautern, Germany: 69-76.
26. Mathiassen, L., *Reflective Systems Development.* Dr. Tech. Thesis, Aalborg University, 1997.
27. McIlroy, M. D. Mass Produced Software Components. In: Naur, P., & Randell, B. (eds*.) Software Engineering: Report on a Conference sponsored by the NATO Science Committee,* Brussels Scientific Affairs Division, NATO, 1969, pp. 151-155.
28. Nygaard, K. Program Development as a Social Activity. in: Kugler, H.J. (Eds.) *Information Processing 86.* IFIP 1986, 189-198.
29. Osterweil, L. Software Processes are Software Too. In: *Proceedings of the 9th International Conference on Software Engineering, 1987,* 2-13.
30. Paesch B. Informatik Spectrum.
31. Paulk, M. C., Curtis, B., Chrissis, M.B.,Weber, C.V. Capability Maturity Model for Software, Version 1.1. *IEEE Software*, Juli 1993, S. 18-27.
32. Potts, C., Catledge, L. Collaborative Conceptual Design: A Large Software Project Case Study *Computer Supported Cooperative Work, Special Issue on Studies of Cooperative Design*. Vol. 5, No.4, 1996: 415-445.
33. Robertson, T. Embodied Action in Time and Place: The Cooperative Design of a Multimedia, Educational Computer Game *Computer Supported Cooperative Work, Special Issue on Studies of Cooperative Design*. Vol. 5, No.4, 1996: 341-367.
34. Schmidt, K., Sharrock, W. (guest editors) *Computer Supported Cooperative Work, Special Issue on Studies of Cooperative Design*. Vol. 5, No.4, 1996.
35. Schön, D. A. *The Reflective Practitioner. How Professionals Think in Action*. Basic Books 1983.

36. Suchman, L., Trigg, R. Artificial Intelligence as craftwork. In: Chaiklin, S., Lave, J. *Understanding Practices – Perspectives on Activity and Context*. Cambridge University Press, NY 1993: 144-178.
37. Tellioglu, H., Wagner, I. Negotiating Boundaries. Configuration Management in Software Development Teams. *Computer Supported Cooperative Work,* Vol. 6 (1997): 251-274.

Localizing Self on Internet: Designing for 'Genius Loci' in a Global Context

Sara Eriksén

In this position paper, I briefly present some experiences and questions I have concerning the importance of addressing the concrete contextual aspects of the use situation, the 'this is me, here and now, and where-do-I-go-from-here' issues of IT in use. In view of the on-going rapid and massive development of integrated Internet/intranet applications, this 'I, me, myself' could be almost anyone, almost anywhere, any time, about whose actions we can conceivably say that they are intentional, but about whose intentions we can only speculate. These are issues that are becoming increasingly complex. In an on-going research and development project (One-Stop Service), we are using a participatory design approach for developing standards for marking, structuring and linking public service information which will allow Internet access to national, regional and local information sources in ways which take into account where the question is coming from and some of the explicit intentions of the individual user. We may not as yet have found any lasting solutions, but we are exploring questions which bring into focus the importance of considering the specificities of human intentional action, and of taking seriously the 'I, me, myself' - and 'we' - of the subject with an objective.

What? Whom? Who is asking? Why? Where are you? What's your problem? In fact, what is it you want, anyway?

The realization that trying to adequately address and understand the contextual aspects of the actual use situation are important for the design of usable software is not a new one. The need for understanding more about the user and the use situation has lead to the development of what are, by now, fairly established traditions in software engineering science and practice, such as Participatory Design and User Centered Design. And more than ten years ago, Lucy Suchman juxtapositioned plans and situated action and examined IT design in the light of the largely unexplored relationship and tension between them (Suchman, 1987). However, the rapid development during the 1990:s of Internet and the World Wide Web has in turn lead to an explosional development of integrated Internet/intranet applications. The diversity of what applications and information can be accessed, as well as who can access what, when, from where and for what purpose, is changing the world of software practice more than we probably have realized. To speak of 'the use situation' or 'the user', as though such an entity could be envisioned in one basic scenario with a limited number of variations, is in most cases unrealistic. Yet the importance of addressing the concrete contextual aspects of the use situation, the 'this is me, here and now, and where-do-I-go-from-here' issues of IT in use, is as great as it's ever been, if not greater. In view of the on-going rapid and massive development of integrated Internet/intranet applications, this hyphenated 'I, me, myself' could be almost anyone, almost anywhere, any time, about whose actions we can conceivably say that they are intentional, but about whose intentions we can only speculate.

The 'I, me, myself' self I refer to here, within quotation marks, is no more subtle, or less complex, a conceptual construction than the individual user using the individual IT artifact. It is the 'I' who may well get up in a moment or two and walk away/move to a different application in disgust, saying something under their breath like 'I can't get anything useful out of this stupid thing!' 'I' is a specific person, in a specific place, at a specific time, with an intention of accomplishing something through an action of 'my' own. What I'm trying to do, here, with the hyphenated 'I', 'me', 'myself', 'my', is to focus very intently, not only on the situatedness, but also on the constructive subjectiveness, of intentional action. The problem, for the designers of modern IT, is at once simple and incredibly complex. We are faced with having to seriously consider the specificities of human intentional action, and take seriously the 'I, me, myself' – and of the subject with an objective. Yet, at least since the days of Hobbes, we haven't been developing very good tools, metaphors and methods for managing the diversities and ambiguities that this multi-perspective approach assumes and opens up towards.

What I'm doing and where I am now

The focus in my research is, broadly speaking, on how modern information technology can be designed to successfully support 'knowing in action'. Using ethnographic field methods, including video recording and interaction analysis, I have studied cooperation and sharing of experience and knowledge in and through work practices in public service front offices (one-stop shops). During the past few years, with the rapid expansion of the use of Internet/intranet solutions within public administration, the IT design issues I have been studying have come to encompass the on-going integration of traditional administrative support systems with public electronic information systems. The 'users' for whom these integrated systems are now being developed are not only employees within public service administration, who know fairly well what they need to get done with the help of 'the system'. A 'user', today, may be any one of thousands of citizens within the municipality, or tourists, or other visitors, searching for information, or in need of help of some kind. The physical access points from which the network can be reached are no longer a limited number of computers on desktops in specific workplaces within one specific organization. Nor is the diversity and heterogeneity any less, looking in the other direction, at the information sources which can be accessed via the network.

When I started my first research project in 1992, my plan was to study and participate in the design and development of computer support for front office work. I set out to focus on the use, design and continual support and development of computer support in one-stop shops. These are front offices for public service administration, offices to which citizens can turn for help concerning many issues which are handled by the municipality (schools, daycare, care for the elderly etc.), the police (lost and found items, the issuing of various types of special permission, passports etc.) and other local and regional authorities.

When I started my research, one-stop shops were a new phenomenon in Sweden. I was especially intrigued by the challenge they posed to traditional bureaucratic information systems in use in public administration at the time. Many of these systems were main-frame based systems, originating from the 1970:s and designed for a different organizational context than the lean, decentralized administration of the 1990:s. What the generalists – the people working in the front office – seemed to need was an integrated help-desk system, through which they could access information from the different specialized systems which were in use in the back offices. The vision which we were working towards at the time was, that by adding their own specific 'generalist' information to this integrated system, the generalists could enhance not only their own work, and that of the front office work team. They would be contributing to a new, over-all information system for the whole organization, bringing in a customer or citizen-oriented view which would serve to revitalize public service administration as a whole.

During the past few years, with the rapid expansion of the use of Internet/intranet solutions within public administration, this vision has come to encompass the on-going integration of the traditional administrative support systems with public electronic information systems. At the same time, and partly because of this very development, there is a growing awareness of how new information technology appears to be developing faster than the models, metaphors and methods used for conceptualizing the sharing and managing of information in organizations, in communities and in society in general.

In my case studies, I initially had the ambition to follow and take an active part in the design of new front office computer support. However, as this design process was repeatedly delayed and postponed, I had to put aside my plans of studying the implementation of designed integration of different specialized systems. Instead, I began to look more closely at how the existing information technology was put to use in the work place.

What I found was, that much of the work that was getting done at the front desks of one-stop shops was accomplished through the skilful managing of ***complex situations***. The computer support, however, was obviously designed for managing one stand-alone task at a time, usually in a given sequence of steps which had to be followed in order to accomplish the task at all. There was little or no support for desk-level intentional action, i.e. tools for planning and following through of various one-person or work-group level projects, no communication and feed-back processes besides e-mail, no thought-through support for managing interruptions and moving back and forth between different applications. There seemed to have been no awareness at all about the actual use situation in the design of the applications, which were now being used for front office work.

Thus, I found that I had shifted perspectives, from what I now perceive as a practice-oriented information flow approach (which I hadn't even realized I was caught up in), to a more holistic focus on work practice. And what I seemed to be catching sight of was, that the way we are utilizing information technology today does not succeed in supporting the everyday work practices through which organizations accomplish their work.

In an on-going research and development project (One-Stop Service), we are using a participatory design approach for developing standards for marking, structuring and linking public service information at different levels. The aim is to allow comprehensive, easy and useful Internet access to national, regional and local information sources in ways, which take into account where the question is coming from and some of the explicit intentions of the individual user. Whatever we develop will have to be of real, everyday use to employees within public service administration at all levels, not only to 'citizens in general'. This is a basic condition for at least two reasons; to guarantee in-depth quality in the development of the decentralized, linked-together information structure, and to guarantee continuing quality, in use and life-cycle design and development, of public service information and inter-active services on-line. Because of my experiences from past and on-going parallell research projects focusing on computer support in one-stop shops, a third reason for using participatory design here is equally obvious; in order to, finally, more seriously, begin to realize the visions of integrated, comprehensive computer support for front office work.

What about 'genius loci'? That is a concept which I have borrowed from architecture and Greek mythology in order to visualize the importance of returning from 'out of time, out of space' objectivity to the subjective, messy complexities of everyday life, where specific places and points in time, and people and their individual intentions, make a difference. 'Genius loci' is latin, and means 'the spirit of the place'. As I've heard architects refer to it, it seems to be something important, though hard to put words to, to which good design must be made accountable.

Brief Bibliography and Contextual information

I am at present a lecturer in the department of Human Work Science, at the University of Karlskrona/Ronneby in southern Sweden. Here, I teach within an interdisciplinary Masters program called *People, Computers and Work* (*MDA* is the Swedish acronym), combining Computer Science and Human Work Science in educating systems developers for the future. I am currently involved in two research projects focusing on the use, design and continual support and development of computer support for public administration in one-stop shops, and the on-going integration of such systems with public electronic information systems. One of these research projects is being financed by the Swedish Council for Work Life Research (RALF), the other by the National Board for Industrial and Technical Development (NUTEK). During the past three years, I have also participated in the EC project ATTACH (Advanced Trans-European Telematics Applications for Community Help, UR 1001, 1996-98), in which the University of Karlskrona/Ronneby was a partner. I am at present involved in the UK-/Nordic Initiative on Information and Communication Technologies (ICTs), which is being sponsored by the Research Councils of Sweden, Finland, Denmark, Norway and the UK (Eriksén, 1999).

The University of Karlskrona/Ronneby was founded in 1989. It is a young and small, but rapidly expanding university, with approx. 3,000 students and 330 employees. The main emphasis in both research and teaching is on ***IT in use*** **–** i.e. on information technology and how it is used.

References

Eriksén, Sara (1998): *Knowing and the Art of IT Management. An inquiry into work practices in one-stop shops*. Ph.D. thesis, Lund, Sweden: Department of Informatics, Lund University.

Eriksén, Sara (1999): *Globalization and 'Genius Loci'; From Intentional Spaces to Purposeful Places.* Working paper for UK-/Nordic workshop held in Ronneby April 15th-16th 1999 on the theme 'Social Theory and ICT'. Website http://www.brunel.ac.uk/research/virtsoc/nordic/eriksen.htm (99-07-15).

Suchman, Lucy (1987*): Plans and Situated Actions. The Problem of Human-Machine Communication.* New York: Cambridge University Press.

Locales Framework as a Shared Abstraction for Understanding and Designing CSCW

Geraldine Fitzpatrick

The Understanding & Designing Gap

CSCW as a field is concerned with the wicked problem of how to improve the way in which people work together through the use of computer-based support. The activities of understanding and designing are intertwined, and integral to both the definition and solution of this problem.

As a multi-disciplinary community, those in CSCW concerned with understanding and those concerned with designing often come from distinctly different backgrounds, neither of which fully prepares their proponents to work at the conjunction of the two areas.

Computer scientists have found that traditional approaches to requirements analysis and design are not able to account for the contingent complexity of lived cooperative work. Also, while systems designers might want to support the 'social' insights that emerge from workplace studies, they are not trained to make sense of sociological accounts of work, and they struggle with how to actually translate social insights into the substance of design [8].

On the other hand, many social scientists, be they from ethnomethodology, distributed cognition, activity theory, and so on, struggle with how to extend or evolve their traditional approaches to meet the demands of systems design, e.g., [10] - demands that their approaches were never intended to address.

A fundamental source of many of the tensions between understanding and designing has been *the role of abstractions* or models in the different world-views. Button and Dourish [1] suggest that computer scientists view abstractions as *generative* in that they give rise to as well as characterise system action whereas ethnomethodologists (and perhaps other social scientists) view abstractions as *analytic* explanations or descriptions of social action.

For our purposes, the point is not to engage in a discussion of the generative/analytical duality. Rather, we want to develop a viewpoint that can be used both analytically and generatively by *all* stakeholders, aiding the communication of workplace and work-practice information to designers, and design information to the workplace.

This is challenging. While the problem of how to improve the way in which people work together through the use of computer-based support is wicked, the form of support to be designed needs to be stated as a tame problem if it is to be built as software and hardware. The *challenge* then is to find an abstraction that accounts for and is grounded in the sociality of work and the uniqueness of each workplace setting, while also accounting for the pragmatic needs of design and systems building. The Locales Framework is put forward as one approach.

Locales Framework

The Locales Framework was evolved out of the broader CSCW community experiences with understanding and designing, and our own attempts at understanding and designing as reported in [3] [4]. We also draw upon notions from Strauss' theory of action [12], namely that actions and interactions are continually permuting, and that social worlds, as groups of people who share some commitment to collective action, provide the major conditions for interaction.

Locale as the Unit of Analysis & Design: The primary unit of analysis and design in the Locales Framework is *locale*. Locale is constituted in the *relationship* between a particular social world and its interactional needs, and the 'site and means' its members use to meet those needs, i.e., the space together with the resources available there. As such, the framework is based on a metaphor of place rather than space [4] [5].

It is because locale arises in this relationship between a social world and its use of space as place that makes it suitable as a shared abstraction, albeit used in different ways. For *understanding*, the emphasis is on the appropriation and evolution of 'site and means' as place for the practical accomplishment of work. For *designing*, the emphasis in on the 'site and means' that are used to meet those needs, and on how better 'sites and means' or affordances could be built for meeting interactional needs.

Locales Framework Aspects: Based on organising principles of perspectives and centres, the Locales Framework is composed of five aspects, all highly interdependent and overlapping. Each of these aspects characterises the nature of work from a different point of view. We explore the framework in detail in [2], suggesting various properties of the aspects and the dimensions along which those properties can be characterised. Here we give a brief overview of each aspect:

Locale Foundations: We first identify the group or social world of concern, looking at features such as how membership is defined, what are the internal structures within the group, and so on. We then identify their locale as constituted by the spaces, objects, tools and resources they use to support their interactions. Locale is the foundation for the rest of the framework in so far as the features identified here facilitate or constrain the life of the social world, as uncovered by the remaining aspects.

Civic Structures: This is the broader context in which the social world and its locales exist. In particular we consider the external influences on a locale, locale lifecycle processes, how locales are structured and related, and how interaction between locales is supported.

Individual Views: This accounts for the fact that groups are made up of idiosyncratic individuals who each belong to multiple social worlds simultaneously. This aspect considers the different views that different members can hold over the one locale. It also considers the different locale views that an individual manages and negotiates simultaneously over the multiple locales in which they participate at dynamically varying levels of intensity.

Interaction Trajectory: This accounts for the social world and locale 'in action' over time: past, present and future; cycles, rhythms and phases; the practical accomplishment of work; routines, articulation, contingencies and breakdowns; information flows; workflows; and so on.

Mutuality: Mutuality is the glue of collaborative activity – how *presence* is enabled in a locale and how *awareness* of that presence is supported. Mutuality enables the 'w' questions to be answered, e.g., who, what, when, where, why, and (almost a 'w') how.

Using the Framework: The Locales Framework can play a communication role between understanding and designing in the following ways:

Understanding: The Locales Framework can be a starting point to help sensitise the analyst and designer to the key elements of a collaborative work situation. In exploring the situation first hand, the Framework aspects can be used to motivate initial questions and observations. In working with pre-existing study data, the Framework aspects can also be used as a basis for structuring the data. Ultimately however, understanding will be uniquely driven by the domain under study.

Designing: Designers can then use the framework aspects to help identify where features can be added to enhance existing locales, or where new locales can be created to coalesce the distributed life of an existing group or facilitate the emergence of a new social world. These enhanced or constructed locales may be physical, or computer-based or a mix of both. The goal is 'technology in context' where decisions about computer-based support are driven by interactional needs, in the total context of how that interaction happens, rather than by a priori assumptions about a representational metaphor at the interface or indeed about the need for a technical solution.

From a *methodological viewpoint*, the Locales Framework can be used in a multi-phase approach, starting with analysis of existing locales of work, and shifting focus to design of new or enhanced locales, iterating and interleaving activities as understanding of the wicked problem emerges.

Thus far, we have used the framework in a number of workplace studies, some leading to design solutions; and for informing the design of a prototype collaboration system called Orbit. See, e.g., [2] [6] [9] for more discussion.

In Summary: The Locales Framework can potentially provide for the following: 1. A common language for understanding and design; 2. Opening up qualitative approaches for people who do not have a social science background; 3. Potential consideration of issues from group to individual, local setting to global context, micro scale to macro scale, and structure to process; 4. A focus on the use of space and its affordances for support of interaction, space and its affordances being the very supports that CSCW is concerned with building; 5. Applicability to a wide variety of domains, physical, virtual, or mixed; 6. A common approach for describing what is as well as envisioning what could be; 7. A framework rich enough to point to key elements of a collaborative environment but sufficiently high-level, open and incomplete so as not to prescribe nor circumscribe all that is of interest.

Conclusion

Does the framework capture the 'right' aspects or the only possible aspects? We expect not. Can the framework really meet the shared abstraction challenges laid out earlier? From our experiences to date, not fully. However, in the nature of all wicked problems, we believe that what we can learn from using the framework will help evolve our understanding of the problem itself, leading to an improved framework or better different solutions.

References

1. G. Button and P. Dourish, "Technomethodology: Paradoxes and Possibilities," in ACM Proceedings of CHI '96, Vancouver, 1996, pp. 19-26.
2. G. Fitzpatrick, "The Locales Framework: Understanding and Designing for Cooperative *Dept of Computer Science & Electrical Engineering*, The University of Queensland, 1998.
3. G. Fitzpatrick, S. Kaplan, and B. Tolone, "Work, Locales and Distributed Social Worlds," in Proceedings of ECSCW'95, Stockholm, Sweden, 1995, pp. 1-16.
4. G. Fitzpatrick, S. M. Kaplan, and T. Mansfield, "Physical spaces, virtual places and social worlds: A study of work in the virtual.," in ACM Proc of CSCW'96, Boston, MA, 1996, pp. 334-343.
5. S. Harrison and P. Dourish, "Re-Place-ing Space: The Roles of Place and Space in Collaborative Systems," in Proc of CSCW'96, Boston, MA, 1996, pp. 67-76.
6. S. Kaplan and G. Fitzpatrick, "Designing Support for Remote Intensive-Care Telehealth using the Locales Framework," in ACM Proc of DIS'97, Amsterdam, The Netherlands, 1997.
7. S. Kaplan, G. Fitzpatrick, T. Mansfield, and B. Tolone, "MUDdling Through," in IEEE Proc of HICSS-30, Hawaii, 1997, pp. 359-548.
8. S. Kaplan, W. Tolone, D. Bogia, and C. Bignoli, "Flexible, Active Support for Collaborative Work with ConversationBuilder," in ACM Proc of CSCW'92, Toronto, Canada, 1992, pp. 378-385.
9. T. Mansfield, S. Kaplan, G. Fitzpatrick, T. Phelps, M. Fitzpatrick, and R. Taylor, "Evolving Orbit: A Progress Report on Building Locales," in ACM Proceedings of Group '97, Phoenix, Arizona, 1997, pp. 241-250.
10. L. Plowman, Y. Rogers, and M. Ramage, "What are Workplace Studies For?" in Proceedings of ECSCW '95, Stockholm, Sweden, 1995, pp. 309-324.
11. H. Rittel and M. Webber, "Dilemmas in a General Theory of Planning," *Policy Studies*, vol. 4, pp. 155-169, 1973.
12. A. Strauss, *Continual Permutations of Action*. New York: Aldine de Gruyter, 1993.

Position Paper for the Seminar Social Thinking –Software Practice

Christiane Floyd

Introduction

In this position paper, I would like to sketch my recent ideas, so far published in German only, on understanding informatics as a discipline and on computer based systems in their interrelation with human action. I start from two questions: What does it mean for someone to be engaged in informatics? How does informatics interfere with reality? I relate these questions on one hand to the discipline (as a whole!), its world view, its basic paradigms and the role models it suggests for practitioners, and, on the other hand, to the products and the assumptions and values they reify.

This work has been motivated by ongoing discussions on the nature of our discipline, its object, its scientific core, its methods and its future development. The sheer diversity of names (computer science, computing science, informatics, datalogi) as well as the separation of concerns in some countries, for example in computer science and information science, points to the fact that there are various complementary or even conflicting views. Viewing the discipline as formal science or as engineering discipline, seeing ist main concern with systems, with design or with media, suggests different questions and avenues of research and poses the problem of what actually ties the discipline together.

The stand I take here is that informatics is concerned with *autooperational form*. That means (1) making operational form in the natural or social world explicit and (2) making autooperational artefacts available for human use. I call the method associated with informatics *operational (re-)construction*. That is, I claim that when engaged in informatics, we have to perceive the area of interest in terms of operational form, and when we make computer artefacts available for human use we create conditions for autooperational artefacts to interact with situated action.

Actions and Operations

In order to do this I have used the concepts of *action* and *operation* for a contrastive analysis of human beings and computers (a term borrowed from my former colleague Walter Volpert from the Technical University of Berlin). The need for concepts that allow to relate human activities and computer functions first came up in connection with establishing *requirements* for software systems, where we found it helpful to set up and relate three sets of concepts, referring to

- the *organization* (in terms of tasks and functional roles),
- the *people* involved (in terms of their activity),
- the *computer* (in terms of its functions).

Somehow, a common denominator was needed to demonstrate how computers could take on some of the work previously carried out by humans, what would remain and what would be changed.

More generally, I see the need for understanding these similarities and differences in all issues of design of computer-based systems, where we constantly rely on decisions on how human beings should interact with computers.

I reserve the concept of action exclusively for humans, and I believe, I use it essentially like Lucy Suchman and in a way compatible with activity theory, sometimes emphasizing the situatedness and uniqueness of human action.

On the other hand, I use the concept of *operation* as a key to understanding what can be computerized. The interesting feature of the concept of operation for my purpose, is that it is well established in different contexts, all relevant for my area of concern:

in organized human activity (ranging from the military, to medicine, to enterprises of different sorts), it refers to proceeding in the situation based on planning with a specific objective;

in activity theory, it refers to the formation of quasi-automated routine on the basis of experience with actions in a domain;

in mathematics, it denotes a computation step in a formalized and interpretation-free manner;

in technology it refers both to the working of machines and to making a machine work.

Crucial for operations is that they are *rooted in repeated human action* in a specific domain leading to routine, and that they are inherently connected with the separation of *description* (representation, specification, planning etc.) and *execution*.

Operations are *described by an observer*. Through description, operations are separable from the individual human beings acting in situations in a unique way. Descriptions are made for a purpose and can be used for teaching, planning and coordinating human activity in connection with specified operations. I hold that this even applies to the concept of "operation" in activity theory which seems to me an analytical concept. Although we can say in general terms "Through experience in action people form operations", an observer is needed in order to give any such operation a name, characterize it and distinguish it from other operations or actions. And such a generalized understanding is used in organized forms of teaching when familiarizing novices with the field of activity of their choice, thus enticing them to form relevant operations through experience with action.

Operations are executed by an actor or agent, for example an individual human being or a group or, in certain circumstances, a machine.

Operational Form

I use the term operational form to denote the result of characterizing and connecting individual operations in terms of temporal, logical and causal relationships. This connection again is made by an observer. Operational form can itself be considered an operation on a higher level. The emergence of operational form is essential for all organized human activity. In the framework of this position paper I cannot go into any kind of adequate discussion of the theoretical foundations for this, but at least I would like to refer to the ideas of Gregory Bateson on the complementarity of (individual) process and form (based on a class of processes) that he sees as fundamental to human learning.

With respect to the present discussion, three aspects of operational form seem relevant:

The relation of operational form and situated action: Predefined operational form is *interpreted* in situated action, thereby bridging the gap between situational needs and specified objectives and constraints. Also, operational form, is constantly modified and developed further through human action. The more generally applicable operational form is intended to be, the more will its description need to be unambiguous and interpretation-free, and the more emphasis is likely to be placed on making it unmodifiable.

The possibility of delegating the execution of operational form from human actors to non-human agents: This is inherently connected with describing operations in an interpretation-free manner by specifying only their formal properties (such as input and output, rules for performing the operation, ...), unlike human actors, non-human agents do not interpret operational form.

The connection of operational form to symbolic and technical artifacts: Symbolic artifacts *reifying descriptions*, technical artifacts *performing execution*, one distinct feature of a software-artifact is that it combines these two dimensions in a unique way (I refer to R. Keil-Slawik and Y. Dittrich for their contributions in this area).

To summarize, I suggest that the computer as a technical agent can replace a human being in terms of operations, not in terms of actions. The question then is, what is new about a computer as opposed to other machines executing operational form (some authors, like R. Budde and H. Züllighoven, see no sharp distinction).

Autooperational form

The term "autooperational" was first suggested by F.-M. Reisin in her studies of the distinguishing characteristics of software-objects. It is formed in analogy to the German term "Automobil" for "car", denoting a technical device moving "on its own" – of course, as specified by human building plans, as triggered by human operation and energized by gasoline. The suggestion is that the computer is a device operating "on its own" – of course, as specified by programs, as triggered by human action and energized by electricity.

The term “autooperational form”, then, points to the potential of certain kinds of operational form to be executed by a computer “on its own”. To appreciate the step beyond traditional operational form that is required, I like to refer to the notion of a “symbolic machine” proposed by Sybille Krämer. She specifies three prerequisites: representation in writing, schematization, and freedom of interpretation as the basis for symbolic machines, this notion being quite close to that of a calculus on one hand and to an algorithm on the other.

In my notion the crucial step to autooperational form is the *modelling of operational form*, thereby making operational form explicit, separating it from its context, and specifying it in a way that allows for the execution by a computer artefact. I call the specific kind of modelling occurring in the context of computing “operational (re-

Operational (re-)construction

Using this term, I wish to point out that

1. In connection with computing, we look at, whatever we look at, *in terms of operational form*. Operational form can be of different kinds. For example, it can be *procedural*, suggesting standard ways of going about connected activities. Or it can provide a *repertoire* of connected operations to be combined in use in flexible ways. Or it can be defined in terms of *constraints* on the outcome and leave the actions themselves open. These different ways of looking at operational form correspond well to programming paradigms (flow-oriented, object-oriented and declarative) and design philosophies.
2. Modelling, a constructive process, is inherent in computing. This holds whether or not we believe in building an explicit formal model as te basis for programming. If we don't, the program itself is a computational model of the area of interest.
3. We may both strive to reconstruct existing operational form or two invent new operational form. On the other hand, whether or not you consider all human knowing to be constuctivist is left to your discretion.

In order to characterize operational (re-)construction, I have coined three terms which are awful, even in German, and do not get better, when I attempt to translate them:

Informatizing, i.e. that we decide to consider something “as an information”, “something” may, for example, be a feature of the natural, material or social world, an event, considering something as an information means to make explicit what so far has been implied, to replace sensual bodily dealing with things or events by abstract intellectual ones, to undergo a process of negotiation leading to a decision.

Discretizing, i.e. that we strive to describe the matters at hand in discrete terms, introducing distinctions and separations; in my view, this is not only necessitated by technical properties of the computer (digital rather than analogue), ist is moreover an inherent facet of the modelling we perform, for example we distinguish and name cases in the social world, we separate time intervals, we use grids to discretize space, etc.

Systemizing, i.e. we consider part of the real world as a system (much in the sense of K. Nygaard) distinguishing and relating two levels: the computer system consisting of hardware and software and the referent system, that part of the technical, natural or social world, that is modelled, affected, manipulated, simulated by the computer.

While, in my opinion, forming a system of discrete, interacting informational elements and relating this to our life world in some way, HOW we form the Computer System, WHETHER we consider the social world as a system, and if so, HOW we constitute the referent system, on one hand accounts for the tremendous variety of existing computer artifacts and on the other hand is the core issue in all questions concerning design.

Autooperational form and situated action

Coming back to discussing the interrelation of computer artefacts and human activity, I consider autooperational form as a specific variety of operational form, which is, inevitably, interpreted in human action, leading to creative forms of computer use, for example "work-
sense of Les Gasser. The role of computers as artifacts in human use is superbly described by Susanne Bødker who views them as mediating human activity. On the other hand, her description in my opinion does not bring out the autooperational character of computer artefacts. Of course, one may claim, that this is a question of design. Computer systems should be designed "as tools" for humans, and so on. But, in my opinion, these metaphors are one-sided. I do not wish to go back to out-dated metaphors such as the (old-fashioned) systems-perspective, but I wish to draw the attention to the fact that, through our activities in informatics, we populate the world with a new variety of entities, namely autooperational ones.

Autooperational agents

In my view, computer programs in execution belong to a novel kind of entities. If I were Popperian, I would call for a "fourth world" consisting of human-made artifacts that are autooperational. Their existence raises a host of philosophical questions, often discussed in controversial terms according to whether one "believes" in Artificial Intelligence or not. I do not want to take a speculative stand here, one way or the other. Rather, I want to stay close to software practice. I am convinced, however, that in designing (quite ordinary) software systems from the simple to the very advanced level we constantly (and most of the time implicitly) take stands on these philosophical matters, since, in one way or the other, we embody them in design-decisions.

The basic question here is: what kinds of interaction between human beings do we allow for? We cannot discuss this without taking a stand of our own, without bringing in ourselves. Also, inherently, there are grave ethical matters at stake, since we manipulate scopes for human possibility for taking responsibility in situated action.

Role of Methodology in a Wider Social Practice

Nimal Jayaratna

Introduction

The growth in methodologies has given rise to many research issues: methodology evaluation, models, philosophy, selection etc. This paper will start with a definition of methodologies and discuss the implications of methodology use in practice.

A methodology is simply "an explicit way of structuring one's thinking and actions. Methodologies contain models to help with this structuring and reflect particular perspectives of 'reality' based on a set of philosophical paradigms. A methodology should tell you 'what' steps to take, in 'what' order and 'how' to perform those steps (methods, techniques) but most importantly 'why' those steps should be taken in that order using those techniques." (Jayaratna, 1994). In this sense, the most important role of a methodology is the intellectual questions it raises for structuring the methodology user's thinking process.

If a methodology is to assist **problem solving** then it must address three distinctive areas of activity, namely:

- Problem formulation (critical thinking)
- Solution design (creative thinking)

- Design implementation (practical thinking).

Problem solving by its very nature implies that the structuring of thinking leads to the achievement of a goal i.e. a problem to be identified and solved. If we are to learn about the success of problem solving then we need to evaluate our activities before, during and after intervention (time periods), and focus our evaluation on three significant elements namely the 'problem situation', the problem solving process and the problem solver(s). The latter forms the three elements of the NIMSAD framework (Jayaratna, 1994). Using this framework, we examine the role methodology plays in organisations. They are:

- Methodology as a way of problem understanding
- Methodology as a way of performing problem solving
- Methodology as a way of helping to implement designed ‘systems’
- Methodology as a way of problem formulation
- Methodology as a way of reducing uncertainty

Methodology as a way of controlling costs

- Methodology as a way of controlling projects
- Methodology as a way of controlling people
- Methodology as a way of exercising power
- Methodology as a way of providing security
- Methodology as a way an instrument of learning

NIMSAD Framework

Given that methodology theory and practice differ so much, the effective transformation of situations depends on the clear and unrestricted thinking processes of the problem solvers. In order to consider the effectiveness of a methodology for a particular situation, problem solver(s) need to ask questions about the 'problem situation', the methodology and about themselves. Some of the typical questions are raised below.

The 'problem situation'

- Why transform the situation?
- Who do I take to be legitimate stakeholders and what are the implications of leaving out others?
- What problems am I trying to solve in this situation?
- What are the essential features and critical aspects of the situation?
- What aspects of the situation are to be transformed and why?
- What are the benefits or penalties of transformation for those in the situation?
- How are failures or successes going to be measured, and what implications?

The methodology

- Which methodology is the most appropriate for identifying 'what' and 'why' (problem formulation) and ‘how’ and by ‘whom’ to transform (solution design)?
- What philosophical paradigms are advocated or promoted by the methodology as being relevant to the situation?
- What domain of the 'problem situation' will be covered by the methodology?
- What assumptions are being made by the methodology about the situation? Are they relevant and useful?
- What structure and steps are offered by the methodology?
- What additional methodological steps need to be developed or substituted by those who are planning to use it? And how are these going to be performed?

- What knowledge and skills are demanded by the methodology?
- What rationale is offered for the steps and the structure of the methodology?

The methodology user

- Why am I intervening in this situation?
- Why transform the situation?
- What benefits am I seeking for myself from this intervention?
- What are the benefits or penalties for me from this intervention?
- What criteria do I consider in selecting a methodology for this situation?
- What skills must or should I have in order to be an effective user of the methodology?

Interactional Resources for System Development

Marina Jirotka

Over the last decade a substantial body of ethnographic studies has emerged that analyse the details of work practices and collaborative work in different settings. An important application of this research has focused on how this understanding of collaborative work can be of practical use in various aspects of system design. This paper will discuss some of the issues and problems encountered in these investigations.

In relation to these exercises, researchers have frequently considered how techniques, methods and findings from the social sciences might most usefully become part of the tools available to the software design community. However, various problems have been encountered with respect to the ongoing, organisational constraints both of undertaking such analyses and representing the findings for use in design. Practical issues have been debated about the level of skill needed by the analyst and which skills are needed to undertake these types of studies. In addition, it is claimed the methods themselves take too long and the results too unspecified. More importantly, it has been noted that techniques are required that have predictive force to assess the implications and consequences of the deployment of novel systems, where these methods have been considered as primarily descriptive. Consequently, researchers have been asked to justify the use of these approaches in terms of the effort expended against the benefit to be gained from such studies. The discussion of these practical problems may be premature as there are more foundational issues to be addressed emerging from these studies. The work of ethnographers studying workplaces, organisations and technology has revealed that we still only have a tentative understanding of how technologies are deployed and used in organisations. Moreover, even fundamental notions such as 'information', 'task', 'communication' and 'collaboration' are still under-specified and, some would argue, need respecifying. Consequently, social scientists have emphasised the need for more thorough analyses of communicative and collaborative work practices.

Though there seems to be a gulf between the two communities of social sciences and software developers, it may not preclude the development of social science approaches that can be utilised to advance practical outcomes within software development. Particularly as it is increasingly recognised by software engineers that a sensitivity to work practice is important. The programme of considering social practice has gained influence within software development, even into systems engineering, to the extent that proposals have been put forward to 'industrialise' ethnography. 'Quick and dirty' approaches have been suggested that attempt to 'fit' the ethnographic orientation into the practical time constraints of software development projects.

These approaches also provide guidelines for conducting such studies and for structuring observations in a way that may be relevant for design.

Some researchers offer more foundational ways of conceiving of the relationship between social science approaches and software design. Dourish and Button in what they call 'Technomethodology' have proposed an extensive reconsideration of the design process, even shifting resources within the software development. This research proposes the development of new software architectures and the use of new ideas in programming languages, requiring additional resources for the transparency of systems to be provided by designers. It suggests a reconsideration of the work and resources that software developers need when designing the contents of systems.

Yet in the short term there may be other ways of considering the development of a relationship that proceeds without this fundamental reconsideration of the design process. There may be ways of utilising methods and tools from social sciences that can be deployed within the current framework of system development building upon what software developers currently do. For example, it may be useful to maintain the present distinction between requirements and design, and consider an approach that not only sets out to do the task quickly, or to develop a set of guidelines or checklists, but can provide a range of activities available to developers. Research at Oxford and related institutions has included developing ways in which the techniques and observations of ethnographic workplace studies can be of use in the context of practical, industrial strength requirements engineering methods. Studies have been utilised detailing work practices across a range of domains and different technologies. These studies have focused on the development of new technologies in collaborative settings and considered particular features such as the implications of distributing work across sites on the ways. The studies have investigated how work is coordinated and the ways in which technology mediates communication. Utilising the sequential organisation of talk as a resource for analyses, the studies were able to suggest implications for projected technologies to support or transform the work of these settings. In addition, previous studies have been used as a foundation for undertaking analyses in novel settings where even a general orientation to unpicking issues such as recipient design and peripheral awareness appeared useful.

Perhaps more significantly, these studies revealed the importance and power of video in the presentation of analyses as a resource for innovation and further analysis. Video presentations can reveal details of a work setting to a designer and can provide some of the richness of the work, in a development context where designers often have little access to the domain itself. However, the video work is not simply a presentation of the workplace or of curious behaviour in a setting. Rather, it is a structured and developed analysis that provides insight into work activities, particularly the tacit practices of participants in the workplace. Although the details of work practice are presented to quite a fine level of detail, (such as showing the coordination of activities, the components of talk and visual conduct) even these rely on a set of common sense practices for their production. Video allows others to see these for themselves where such practices may remain unexplicated to a degree for the purposes of design. Hence they act as a resource for design at the very least in conveying issues about the workplace to be considered. The research at Oxford has investigated the use of these ethnographic analyses supported by video in hypothesising about and initiating the design process. In relation to particular projects, both research and industrial, we have examined the implications of proposed new technologies with regard to detailed work activities and how these have been tested in real world settings. The results can imply further analysis of some particular feature of the setting or a reorientation of the proposed design. This work does not mean to preclude the foundational work that still remains in this area. Rather, it suggests practical studies that may provide a useful starting point for deploying ethnographic techniques/social science approaches for software development.

Use of Groupware in a Networked Enterprise

Helge Kahler

Introduction

In this position paper I describe my work in a project dealing with the support of the internal cooperation of small and medium sized networked enterprises. The project contains aspects of research – trying to find out what is going on in organizations – and action – supporting organizations to better cooperate by technical and social measures. The position paper describes the networked enterprise that we mainly work with, our work there and topics related to "Social Thinking – Software Practice".

Sigma

Sigma (all names changed) is a network of about 200 mainly freelancing consultants and trainers who build small or large teams to work on projects. The main areas of Sigma activity are training, e.g. for software packages, and consulting. Projects are acquired by either the freelancers or the Sigma management. Sigma operates nationwide in Germany; the organization members are distributed all over Germany usually working in their »Home Offices«. As a response to Sigma's growth in the past years and to the corresponding new requirements for coordination two forms of subdivisions were chosen, one along business objectives and one according to the location of the members. As Sigma has now more members than could know each other personally, strong personal relationships are established and maintained in local subdivisions. The boundaries between staff who »belong« to Sigma and those who do not are blurred and mainly set by the self-perception of individuals.

SigSys, the low-end groupware system employed by Sigma for purposes of communication and coordination, is an online-offline e-mail and bulletin board system. It can be operated via modem or ISDN. E-Mails can be exchanged within SigSys or to and from the Internet. Individuals may subscribe to bulletin boards which deal with issues pertaining to certain projects or regional groups and post and read messages. Moreover, SigSys permits the sending and receiving of binary data like text documents or overhead presentations that are often exchanged between members of a project team. Any project member or customer who is granted permission by the Sigma management, has access to SigSys. Currently about 190 persons have access to SigSys. Subscribing to special bulletin boards requires extra permission. Compared with an ordinary Internet account the range of functionality of SigSys is relatively small. Yet, beyond its low costs (purchase, service and maintenance) SigSys has particular advantages: SigSys is well suited as an internal medium of communication; it is relatively easy to regulate access rights and restrictions within SigSys; and SigSys is easy to install. At the same time SigSys has various disadvantages: the system has to be maintained centrally; adaptation and tailoring of the system cannot be done by the users themselves; even smaller changes are exclusively carried out by the maintenance personnel at the provider's site.

Three main issues determine the use of SigSys within Sigma:

- although Sigma itself attempts to be very flexible in order to react to market changes and to deal with its partially fuzzy structure, SigSys is a rather rigid system;
- SigSys relies on content provided by its users;
- despite serious disadvantages of SigSys the convention to use SigSys for internal communication and its stable use made SigSys an integral part of Sigma's communicative infrastructure.

Social Thinking – so far

The goal of our InKoNetz project (Integrated Cooperation Management for Networked Organizations) is to find out how the internal cooperation of small and medium sized networked organizations can be supported by technical and organizational means. We strongly believe that it is important to get to know the work of a person and organization if you want to support it. To reach this goal we are conducting studies in several of those organizations, the most interesting of which is Sigma, and we do literature research. In Sigma we took part as participating observers in a workgroup on Sigma's information and knowledge management and we observed the usage of SigSys, following discussions about Sigma-related content as well as meta discussions about SigSys itself. In addition we conducted 16 semi-structured interviews with stakeholders in different positions and places in Sigma. One of the results were suggestions for the design of software supporting the internal cooperation of such organizations (Rittenbruch, Kahler & Cremers 1998).

Software Practice – now what?

SysShack, the software company that makes and sells SigSys is part of the InKoNetz project. They are planning to redesign and reprogram the successor of SigSys from scratch. We have already shared with them our perspective about the way Sigma works and what a software to support Sigma could be like. We are also planning to involve users from Sigma in the development of this new software. However, we face several difficulties (see Rittenbruch & Kahler 1998 for more detail):

SysShack is not very open to include "social thinking" in their development; they basically think they know what they are doing and know what the users need;

Sigma members are distributed all over Germany, so it is difficult to get a group together;

most Sigma members are freelancers, so the participation in SigSys' redevelopment cuts off their own time rather than their employer's as in ordinary participatory design processes.

Under these conditions much of the discussion about the involvement of social theory in software practice seems academic. Several questions arise:

which social theory helps to understand such a team-oriented networked enterprise?

do we not need to involve much more than a social theory about work, namely organizational theory, theory of networks, theory about micropolicy and power structures in groups and organizations and more aspects and perspectives from still other research disciplines?

what is the practical link between social thinking and software practice – how can we put into action the notion that one is linked to the other?

Obviously, there is a need for discussing not only the absolute quality of social theory for software practice but also the question how theoretical insight from different disciplines can be fruitfully bundled and applied to support software practice.

References

Rittenbruch, Markus; Kahler, Helge, Cremers, Armin B. (1998): Supporting Cooperation in a Virtual Organization. In: Hirschheim, Rudy; Newman, Michael; DeGross, Janice I. (Hrsg.): Proceedings of ICIS '98, S. 30–38

Rittenbruch, Markus; Kahler, Helge (1998): Virtual Organizations as a Challenge for UCD. Position Paper at Workshop 12 (Workshop on User Centered Design in Practice – Problems and Possibilities) of the CSCW '98 in Seattle, WA, available at http://hci.cmd.uu.se/~jg/PDC/

Beyond tasks:
Actions, activities, and social aspects of computer use.

Victor Kaptelinin,

Understanding Users

One of the most important issues in establishing the relevance of social sciences to software development is finding out how the insights coming from social sciences can be transformed into design of more useful and usable systems. In this respect, the gap between social sciences and software development, is, basically, the gap between users and developers. Social sciences, with their theories, concepts, and methods, can provide a better understanding of what people actually do when they use technology, what they really want and need. In this paper I claim that a possible obstacle to a more substantial contribution of social sciences to software development is current preoccupation of some of system developers with the concept of task. In my view, a way to bridge the gap between the social and the technical would be to shift the focus from tasks to user actions and activities. (This position has very much in common with the one presented in position paper by Markku Nurminen for this seminar.)

Limitations of Task Analysis

Users' and developers' perspectives on computer technologies are fundamentally different. Users want to achieve some meaningful goals, while designers are mostly focused on technology per se. Efficient design and use of computer technology requires that the differences between users' and developers' perspectives be to some extent reconciled. Users have to formulate their goals according to the affordances and constraints associated with available technologies, while designers should get an insight into users' needs to make a system useful and usable. Therefore, a "middle ground" between means and ends – a level of representation of work at which users and designers speak a common language – should be somehow established to ensure mutual understanding.

Currently the notion of *task* plays a key role in bringing users' and developers' perspectives closer together. This notion is meaningful for both parties. It is not uncommon for users to decompose their work down to the level of specific tasks that can be carried out with the help of a system. As for designers, they can focus on understanding and supporting users' tasks without being lost in the enormous amount of potentially relevant information about how people use, or might use, computer technology.

Undoubtedly, the focus on tasks was an important positive factor that contributed to the impressive achievements in designing new powerful computer tools in recent years. Task analysis has established itself as a popular practical approach to helping designers provide efficient support for users. However, it is becoming more and more obvious that a heavy focus on tasks is too narrow. It can be a limiting factor that prevents designers from a deeper understanding of users' needs and negatively influences further development of information technology.

Of course, task analysis is not a monolithic approach but rather a generic name for a wide variety of tools and methods. Accordingly, the notion of task have a very broad meaning within this approach. However, a closer look at how this term is actually used in specific techniques, like GOMS or KAT, reveals some common features of what is meant by "task" within the task analysis approach: typically tasks are routine, carried out individually, and have pre-defined goals. There is no doubt they constitute a significant part of human-computer interaction and have to be properly understood and supported. It would be a mistake, however, to think that such tasks are the only aspect of human use of computer technology that need to be supported. The reality of computer use cannot be reduced to just an additive sum of routine tasks.

Empirical evidence and theoretical arguments, for instance, within the area of CSCW, call for a major revision of the underlying assumptions of task analysis. Nonetheless, as mentioned above, the notion of task typical of most task analysis techniques does adequately describe some important aspects of human-computer interaction. Therefore, it cannot just be rejected, but must rather be put into a wider conceptual framework. From my point of view, the most promising framework for providing a coherent perspective on such diverse phenomena as routine tasks and situated actions is Activity Theory. This approach, originating from Russian cultural-historical psychology, has recently gained much attention in the field of Human-Computer Interaction.

From Tasks to Activities and Beyond

Like most versions of task analysis, Activity Theory construes computer use in terms of hierarchically organized processes oriented towards certain ends. However, functional subordination of goal-oriented processes is not, according to Activity Theory, the only organizing principle of human activity. First, there are qualitatively different layers of human interaction with the world: activities, actions, and operations. Second, since people at any given moment in time are driven by different, potentially conflicting motives, actions are poly-motivated – that is, they belong to several hierarchical activity structures simultaneously .Third, actions are often distributed between several individuals, so that each person contributes to the collective effort but action as a whole is carried out by the group. Fourth, human actions undergo continuous developmental transformations. Finally, human actions are mediated by socially developed artifacts.

From the perspective of Activity Theory, tasks correspond to lower-level actions. They constitute a significant part of everyday use of computer technology, but they are usually embedded into higher-level actions which require the definition or re-definition of goals and strategies, are influenced by constraints imposed by other objectives, and are performed in co-operation with other people.

Concluding Remarks

In this position paper I claim propose that the framework of activity theory can help establish a dialog between software developers, users, and specialists (both researchers and practitioners) interested in social aspects of the use of technology. The concept of task appears to be the lowest common denominator, a starting point for reaching a broader representation of how a technology is used, that would be meaningful for both system developers, on the one hand, and users (and "social thinkers" as their representatives), on the other.

The limitations of the concept of task require that tasks be analyzed as embedded into higher level units of activity. Individual and collective activities are, in turn, embedded into larger - scale social context and take place in certain social settings. More specific conceptions of how the idea of embedded activity contexts can be used in design and evaluation of software is currently being explored in a number of ongoing projects at the Department of Informatics, Umeå University, Sweden.

Media Qualities that Support Design and Learning Processes

Reinhard Keil-Slawik,

In order to design a learning supportive infrastructure it is necessary to identify which problems of the learning process are technological problems and what the primary technological functions are. I will briefly sketch a theoretical framework which allows me to identify general technological functions needed to be implemented. However, to appropriately determine the specific implementation of these functions empirical investigations are needed, because it is the fine tuning of all variables, of technical means and didactic concepts, innovative ideas and available resources which ultimately determine how to set up an appropriate infrastructure.

Two general ideas may help to illustrate briefly the approach we have taken. The first idea is based on an argument provided by J.J. Gibson. Gibson came to the conclusion that it is human information processing which allows us to distinguish between what is imagined and what is real. When we view a phenomenon from different perspectives, or when we modify what we perceive and interpret the respective changes we gather information through our bodily activities. We cannot, as Gibson claims, gather information about a purely imagined phenomenon, because the result of any purely mental modification or change in perspective is completely controlled by our mind. Thus, no difference between what is imagined and what is real can be discovered. In order to carry this argument a step further we can say that nothing can be learnt about a certain phenomenon if we are unable to generate and interpret different patterns of perceivable responses. Thus, learning requires feedback either by personal observation or social response; in general it is a combination of both. Thus, in order to gain information about a certain phenomenon we have to create an environment that can be perceived and manipulated.

The second idea is based on the fact, that the biological (genetic) basis for cognition has not changed over at least the last several thousand years of human evolution. Our cultural accomplishments do not reflect an improvement in our biological capacity of intelligent behaviour but rather our ability to create ever better expressive means to support productive thinking (Leroi-Gourhan) and to create a socio-symbolic universe (Elias) which gives us the ability to build our learning activities on the ideas and accomplishments of our. Thus, with respect to the evolution of culture in the large as well as with respect to the evolution of individual mental capacity it can be said that learning is an inherently social activity which is based on the use of physical artifacts. However, the use of symbolic artifacts requires technology. The crucial question then is: What are the specific technological functions which provide an adequate support for learning?

In order to answer this question, I start with a distinction between the terms memory and storage. Human memory is dynamic, irreversible (one cannot deliberately forget or erase something), selective and relative to individual experiences and views. Storage are repositories which usually serve to place physical items in locations and to retrieve them later. The functions of storing and retrieving should not modify the items to be stored (authenticity). In this view, the human memory processes information whereas computers store and retrieve data.

Aside from the human brain and its memory functions we talk *about external memories* when the items in a storage are continuously interpreted and modified due to some individual or social intellectual process. A university library, for instance, is not just a storage (repository for books and magazines), but a living body of knowledge for a scientific community which is continuously modified to embody the actual state of knowledge. In this aspect the library serves as an external memory.

The same holds for course material in education. During a course students learn to explore the materials, meaning is associated with formulas and data, assess and discuss the relevance of examples, and add their own material which may be individually produced, copied from some other source or be the result of collaborative efforts. The way how they use the external memory reflects their actual state of knowledge and changes the more they become knowledgeable. At the end of a course, participants have their own personal understanding, but they will also share a common understanding of the material to the extent of their active involvement in its collaborative use and redesign; a somewhat arbitrary repository of semiotic artifacts has evolved into an external memory.

Now, a storage is an indispensable part of an external memory. Items within a storage can be created, moved, arranged or deleted to express people's understanding and to support their mental activities. The actual physical state of an external memory, however, can only embody this knowledge to a limited extent, because it does not give a detailed account of its evolution and why it evolved that way. Furthermore, technological support should not aim to prescribe the way in which the storage is used, but rather provide primary technological functions such as:

CREATE: Making the phenomena to be studied perceptible and manipulative through e. g. techniques of visualization, symbolization, or simulation.

ARRANGE: Arranging items (simultaneously) within the perceptual field in such a way that they can easily be related to each other according to the problem at hand.

STORE: Providing ways to embody with physical means what is perceived as belonging together or forming a meaningful whole (e. g. by physical connection, spatial arrangements, or physical links).

These functions are the basis of learning supportive infrastructures. Their implementation helps us to cut down on the resources needed for the provision and maintenance of educational material. Consequently, the implementation strategy is to reduce discontinuities in media use by integrating different media types and tools and by making them accessible wherever learning takes place. Multimedia and web-technologies are today's most effective means to implement these primary functions.

Primary functions are essential means for the creation and maintenance of external memories insofar as they should allow students and teachers to create and arrange any sort of educational material in any way without enforcing specific pedagogical approaches or didactic concepts. This gives a high degree of flexibility, but it may not provide sufficient support. A textbook or CD-ROM is not just an arbitrary physical arrangement of different media types such as text, graphics or video, but it is specially selected and designed material in the form of, for instance, introductions, examples, exercises, or references for further reading. The arrangement and the selection of the material follows didactic principles and quite often it may be useful to propose a certain sequence of learning activities or guide learners through the material along a pre-specified path. We call these technological functions which guide users or enforce specific sequences of activities according to underlying didactic principles secondary media functions of the external memory, because these functions embody specific (empirical) knowledge about how to design learning environments. Functions that implement a specific didactic model can be grouped into three categories:

STRUCTURE: Providing operations to create and maintain specific arrangements of tools and documents such as, for example, learning units, which may be defined as a compound document containing a specification of the learning goal, an introductory text, some examples, a set of interactive tests, and further references.

INSTRUCT: Developing means for instructional design, especially tutoring and feedback mechanism for self-guided learning.

COOPERATE: Creating environments that support specific methods for cooperative learning and problem solving, such as brainstorming or future workshops.

The implementation of secondary functions requires careful analysis of the specific knowledge to be taught and should be based on theoretical principles to justify the additional resources needed to design the tools and materials. Since secondary functions are tied to a specific didactic approach, they have to be implemented anew for different learning environments.

Finally, attempts can be found in literature to implement tertiary functions, although these approaches have not created very promising results so far. Tertiary functions embody generic knowledge of problem solving and human understanding. The idea is to built smart or intelligent systems that have the ability to understand the users or learn from their behavior such that the systems adapts itself to the respective needs of the specific learner. Tertiary functions are:

ADAPTION: Designing mechanisms which analyze the learner's input and change the system's behaviour due to the result of this ongoing analysis.

INTELLIGENT TUTORING: Developing systems that are able to cope with unanticipated or wrong input by the users in an intelligent way. The system must embody models of the Learner's knowledge, of problem solving knowledge of domain experts and knowledge about intervention strategies of teachers.

NATURAL LANGUAGE PROCESSING: Enhancing the systems ability to "understand” the input of learners by natural language processing mechanisms.

Tertiary functions represent the most ambitious goals in designing learning environments, and it seems doubtful whether this approach will ever lead to practical solutions. (This discussion leads to the general discussion of what can be accomplished with artificial intelligence. A more specific review and assessment of intelligent tutoring systems is provided by Sedlmeier & Wettler (1998).)Secondary functions are often regarded to play a key role for a new generation of learning environment such as computer based training (CBT) or computer aided instruction (CAI). However, implementing secondary functions generally leads to an approach of supporting individual learning and thus, often counterbalances attempts to set up infrastructures that support collaborative learning.

To sum up, it can be said that primary media functions are the domain of engineers, secondary media functions are the domain of teachers and prychologists, and tertiary media functions are a matter of interdisciplinary research reaching far into the future. In a practical development project software engineers should always strive to identify and reliably implement the primary functions in the specific setting. When it comes to secondary functions they should ask for professional help and take an interdisciplinary development approach. The primary task of the engineers is to figure out how primary functions may help to implement secondary functions; based on their technical knowledge and with their limited competence they should not aim to design secondary media functions themselves. Designing secondary functions is a pedagogical problem not a technical one. Tertiary functions are a different kind of story that will require yet another position paper.

The notion of external memory provides the general framework on which further concepts such as the notion of media functions are based. Media functions provide a conceptual means to talk about cost/benefit relations and to describe the role of semiotic machines with respect to teaching and learning in such a way that it helps us to design a learning supportive infrastructure by identifying the role of technology in respect to cognitive processes of creating and using educational materials.

Participatory Design –Research Methods and Research Practices

Finn Kensing

Research on software practices for the purpose of improving these practices benefit from an action research approach. Action research has traditionally been carried out in corporation with groups with scarce resources in relation to the subject matter.

The researches and the groups they work with each have their own agenda. The researchers want to learn about a certain area of interest. In my case: Design and use of computers in specific organisational contexts. I want to explore the ways in which computers relate to the work of employees. I also want to understand the work practices of designers and to develop participatory methods and work practices for designers and users for the analysis of problems and opportunities and for the design of computer artefacts that actually support users in their work. My motivation is that too many computer systems have been introduced based on a too narrow understanding of users' work practices, and that users - given the opportunities - can play an active and constructive role in re-designing their work and artefacts that supports it. I.e. my agenda is to build theories of and methods for design.

Over the last twenty years I have worked with trade unions, groups of users and their managers, and with IT designers and their managers. Their agenda has been to understand certain aspects of their work, to get help in solving specific problems, and in developing technology and work practices enabling them to increase the quality of the products/services they deliver as well as increasing the quality of their working lives.

Action research is strong in proving guidelines for how to set up and maintain relations between researchers and those with whom we work. It provides valuable suggestions as to types of activities for data collection, for taking action to improve the situation at hand, and it provides promising accounts of the ways in which the research improved the (working) life of those the researchers worked with.

However, action research is weak in terms of guidelines for data analysis, for contributing to research in general, and action researchers have been accused of being good politicians but bad researchers. In order to remedy for the first point I have turned to Grounded Theory, which provides a set of guidelines for the generation of theory based upon qualitative data. The second and third claims are related, and in one respect they are due to the situated nature of action research. However, more could be done to compare results across various projects. The purpose of the paper - which this position paper will lead to - is to reflect on three large research efforts that I have been involved in seen in the light of other action research projects on Participatory Design.

In the late 70-ties I was part of the DUE-project (Danish acronym for Democracy, Development and Computers). The project's aim was to develop a strategy for local trade unions for them to influence the ways in which computers were introduced at their workplace. A project group of 3 shop stewards and 4 researchers worked together for 3 years. The activities comprised a survey among local unions about their experiences with computers at their workplace. 3 work groups carried out the main activities. They were comprised of 4-7 workers, 1-2 researchers, and 1-2 master students. Over a period of 1 1/2 year we worked at a shipyard, at a factory and at the Post Giro office. We analysed the use of computers, the ways in which they were introduced at the workplace, consequences for the workers, and we developed strategies for them to gain more control in relation to development and use of computers. The major part of these strategies were adopted by the local unions we had worked with, and to some extend they were taken up by other local unions. The diffusion was furthered by a one-week course designed by the project group as part of the union's training program for workers. The courses were given 4-10 times a year over a period of 10 years. Members of the project group were also involved in setting up local and central technology agreements.

In the 80-ties I was part of the MARS-project (Danish acronym for Methodical Work Practices in Systems Development). The project's aim was to increase our - and systems developers' - understanding of the challenges of their profession and to experiment with new methods for systems development. A group of 3 researchers worked for 3 years with practitioners. In the first phase 4 companies participated, 3 of which continued in the second phase. Work groups were sat up in each company consisting of 4-6 practitioners, 1-2 researcher and 0-2 master students. We analysed the work practices of the developers and arranged experiments with new tools and techniques. We explored new ways of working with users and more efficient ways of managing the systems development process. The participating practitioners gradually integrated the results in their repertoire for action. Based upon our experience the researchers and some of the practitioners wrote a textbook, "Professional Systems Development", which is still used at universities and technical schools, as well as in on the job training of professionals.

Since the late 80-ties two colleagues and I have conducted the MUST-program (Danish acronym for Theories of and Methods for Design). The aim of the program is similar to that of the MARS-project, however focusing on the early activities in systems development, which we name design. We use the term design in the same way as architects do - focusing on the analysis of needs and opportunities, and the preliminary design of functionality and form. The MUST-program comprises 9 design projects, some of which have been carried out by us together with users of the participating organisations while in others we have worked with design practitioners and their users. Lately we worked as coaches for designers using our approach in 3 projects. These projects also comprise studies of the work practices of the designers involved. The purpose of the MUST-program to engage in these projects is for us to test and further develop our approach to design, which we are currently documenting in a textbook.

The Role of Formalisation in Software Construction

Herbert Klaeren

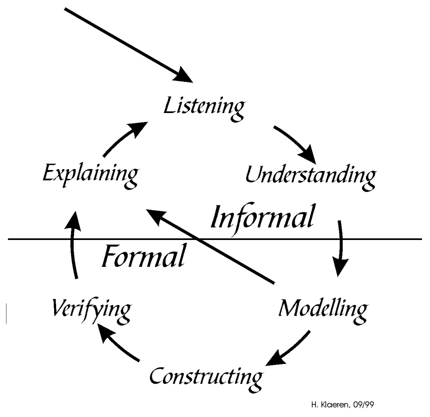
Hercules at the Crossroads

When I came to Tübingen in 1988, people reminded me that Gustav Schwab had lived there around 1800. One of his works was an edition of legends from ancient Greece and Rome *('Sagen des klassischen Altertums')* which I had read when I was a boy. Of course, the stories about Hercules had fascinated me very much at the time, even the one with the moralistic background that is mentioned in the title. Here, young Hercules goes to a lonely place to think about his future when suddenly two goddesses, Vice and Virtue, appear to him. Vice promises him a life of sheer joy and delight, whereas Virtue threatens him with a life full of labour and pain but also posthumous glory. Needless to say that Hercules, being a true hero, follows the advice of Virtue.

What does this mean for software construction? I have started my professional career as a die-hard formalist and theoretically-minded computer scientist. I have at all times believed, and so taught my students, that we can allow ourselves the joy and delight of programming only after we have fought our way through the labour and pain of formal software specification. Here, 'formal' is meant in a really strict, mathematical sense; I’m well aware that in the software business there are also some very weak interpretations of 'formal'.

Orderly software engineering and shrink-wrap software

Of course, recurring to formal software specification reminds of Dijkstra's wall, separating the formal, orderly world of computer profession from the informal, chaotic, so-called 'real' world. From time to time, a customer drops a specification over the wall, and we check it for consistency and completeness. If it passes our test, we use formal methods to derive a correct program according to the specification and some time later we throw a magnificent program back over the wall. As long as we keep inside the wall, nothing can happen to us: we cannot get lost in the real word jungle, we cannot drown in the applications swamp, and we are not bitten by wild animals. Only that we cannot *really* keep inside the wall: after all, our programs are meant for the real world and must prove their value out there. Our customers usually will not be able to understand and enjoy our fine algebraic specifications and certainly they cannot write them, so we must go outside the wall and explain what we've understood and how therefore our final product will behave.



fact, being software engineers, we only can discuss in meaningful way about *formal* objects. I have spent considerable energy in the development of tool support for algebraic specification. The strategy here is: *listen* to your customer, develop an *understanding* of the problem, write a constructive algebraic specification *modelling* what you've understood. Then, have your tool interpret the specification and *explain* its behaviour to your customer. Discuss scenarios together and see whether the formal specification exhibits the expected behaviour. When your customer is (after several cycles) satisfied with the results, go off and *construct* programs and *verify* that they are correct with respect to the specification. Again, *explain* the program to your customer; if you've put sufficient energy into the short cycle, there should be no need to repeat the long cycle.

Yes, I know that this procedure suits only a certain fraction of the systems we have to develop, and that it ignores important ergonomic details such as user interface, organisation of work etc. I still claim that it leads to unbeatable quality where it applies.

In 1994, however, I attended ICSE'17 in Seattle and, among others, participated in a session on shrink wrap software. Here, I heard some puzzling and shocking news that made me think twice about the software development process. They said that the most important task of a software team leader is to get his team into 'ship mode', that time to market is more important than product quality, that the noble principles of software engineering science don't mean anything to the mass market, that you have to be a pioneer who grabs a portion of the market with a possibly faulty product but then emits a steady flow of improved and enhanced versions. In this philosophy, a bug isn't a stain on the professional dignity of the software developer, it is an ephemeral nuisance, bound to disappear in the next release.

Yin and Yang in Computer Science

In the April 1998 issue of CACM, A. C. Sodan reviewed some insights from brain research, related them to the Yin/Yang paradigm of Eastern philosophy and attributed computer science terms to left brain/right brain and Yin/Yang categories. Of course we know that you need both halves of your brain to really master your world; in the same way, Eastern philosophy claims that it needs both Yin and Yang to give a complete picture of the world. The following table is software specific and a shortened version of Sodan's table:

Yang	**Yin**
Plan, concept	Implementation
Abstraction, prototype, class	Concrete example, individual instance
Objects	Relations
Local work	Communication
Theory, formalism	Application, practice
Algorithm	Pattern-controlled or case-based decision
Crisp values, identity	Fuzzy values, similarity
Formulas, verbal description, numbers	Graphical presentation of information
Optimal solution	Heuristic

The dilemma resolved

What does this mean for the value of formal methods in software engineering? One conclusion is, of course, that we cannot restrict ourselves to purely formal work; the other, however, is that we cannot succeed without formal methods. In the virtual reality of software engineering, Hercules can probably walk the way of Virtue (i. e. formal methods) and of Vice (i. e. software hacking) at the same time.

Making Use of Anarchy in Systems development

Ralf Klischewski

This paper presupposes that systems development is most successful when the actors participating are not limited in the way of relating their professional work to all the social aspects relevant for systems development projects. The following is to overcome some limitations of thinking...

1. Software Practice

Software practice embraces technical as well as social aspects. Here, I am concerned with how software projects are related to social aspects of the user's world and not with, e.g., how programmers interact in a team or how social thinking may inspire the art of software development in its technical dimensions (although, it might be worth discussing 'eXtreme Programming' in the context of anarchy...). Thus, I would like to restrict my notion of 'software practice' to activities centred around relating to the application context emphasizing the social interaction, e.g.: 'systems development consists of those activities which aim at changing an organization through the use of computer technology.' (Andersen et al. 1990)

However, talking about words, meanings, and perspectives is the work of science. In practice, the project work needs to be done regardless of any scientific distinction between software, system, or organizational development. Software experts are involved in all of these areas (not the least because of other actors often do not take up responsibility for the aspects of social development). Therefore, reflecting of and educating for software practice, based on scientific analysis, needs a broad view without excluding any perspectives on the social a priori.

2. Social Thinking

Traditionally, the science behind software practice is computer science, the source of solutions for all engineering problems related to the construction of hardware and software. There are other sciences around (e.g., informatics, information science, management science) abstracting somewhat from the technical engineering and putting more emphasis on analyzing use aspects as well as recommending methods and procedures for systems development. In all these sciences, the mainstream acknowledges systems thinking as a kind of meta-theory for systems development. Even the work of Checkland (1981), 'derived from a decade of action research' (cf. book cover of 1989 edition) and being very critical with 'hard systems thinking', holds to the point that there is the real world (practice) and systems thinking (science) – and that it is the task of the developer to establish an adequate relation (e.g., by using soft systems methodology). On the other hand, the social sciences provide us with so many more ways to think about the social – how can we enhance the richness of reflection without weakening the intellectual foundation of systems construction?

3. Position

We need systems thinking to develop any kind of computer supported system (software practice).

We need not only a (pragmatic) relation between system thinking and social practice, provided by methodologies, but also a (theoretical) relation between system thinking and (any other kind of) social thinking, provided by an enriched meta-framework for computer science and/or related sciences.

I suggest to move from a monistic to a dualistic 'Weltanschauung': there are systems – and there is anarchy (see below).

I believe that the intellectual space opening up by relating ‘systems thinking’ and ‘anarchy thinking’ can embrace all or at least most of the dimensions of reflection and threads of discussion already introduced in the debate on social aspects of systems development.

4. Anarchy Thinking

There is a considerable amount of scientific work on anarchy and anarchism. But unlike ‘systems thinking’, the term ‘anarchy thinking’ has not been introduced and discussed in scientific literature (only, there had been a noticeable epistemological debate on the 'anarchistic theory of knowledge' of Feyerabend, 1975). In this paper, I can only outline some key elements of ‘anarchy thinking’ from my point of view (cf. Klischewski 1996):

values and emotions: you love it or you hate it – anarchy is either the social paradise to strive for (e.g., attributed with social equality, total freedom for self-determination, social organization without claims to power), or the social hell on earth to fear (e.g., violence everywhere, total chaos, no responsibilities, etc.). In contrast, reflecting on systems is almost soporific...

subjects and activity: since ‘system thinking’ is abstract (i.e., not bound to any real world domain a priori), the system modeller is free, e.g., to include humans or machines on the same level as instances to carry out some action. In anarchy thinking, only humans act as individual or collective social actors.

individuals and the social: like in a system perspective, anarchy is about the relation of parts (individuals) to a whole (community/society) – unlike any system perspective on human activities, anarchy does not presuppose any 'fixed' (i.e., reliable, maintained) relation between any two 'elements', especially no hierarchical relations.

self-organization: in systems thinking, you can only **observe** a system organizing itself (of course, you may observe yourself taking part in the system) – in anarchy thinking, you can only **do** it, i.e. take part in activities which an observer may identify as organization (of course, you may reflect on your own experiences, individually or collectively).

love and control: anarchy and systems each appeal to different fundamental organizing principles. Systems thinking enables control, i.e., processes transforming the state of any system can be checked whether they are going to maintain the (higher order) structure of the system, thus providing a means to direct process behavior from the observers point of view. In contrast, anarchy thinking usually negates any control but self-control (some anarchist thinkers did incorporate control mechanisms for social organization, e.g. Bakunin). The success of social interaction is relying on mutual trust and love – an organizing principle which, without further support, moves anarchy in the realm of Utopia. On the other hand, however, there is no viable social community solely based on control.

The above issues are related to phenomena of 'the real world'. In addition, these ways of thinking differ in their epistemological background: General systems theory (Bertalanffy) is based on mathematical assumptions, and the main applications and contributions of 'systems thinking' can be found in the natural and engineering sciences. 'Anarchy thinking' is based social experience (the literature is mainly referring to events in the periods of the 19th century worker's movement, the Russian revolution, or the civil war in Spain), it is well in line with historical approaches and critical theory.

At the end of his book 'Systems Thinking, Systems Practice' Checkland (1981) writes several pages exploring the relation of Soft Systems Methodology and the Frankfurt School/critical theory, pointing out a 'significant compatibility' between the two. But, turned of by the 'apparently wilful obscurity of much of the writing of that School' (p. 11), he decides to stick with systems thinking as the foundation for scientific reflections and to incorporate the dealing with the richness of the social into the methodology.

This approach has contributed very much to theory and practice of systems development. However, at the same time it turns out to be a rescue operation for the monistic "Weltanschauung" of computer science and other sciences relevant for systems development. Systems development projects interfere not only with parts (e.g., individuals) but with the whole of social organizations (see definition above). Systems thinking is one important and very useful approach to reflect on these aspects, but it is not sufficient: there are many phenomena within social organizations influencing systems development projects which cannot be understood within the framework of system thinking.

Trying to deal scientifically with the 'social whole' as an entity is an extremely difficult subject. There are so many traps leading to totalitarism or obscurity, some schools of thought even ban any holistic approach beyond the realm of scientific rationality. Indeed, ultimately there is always some mystery about 'the whole' as there will always be a mystery (such as love) in human interaction. Sciences of the past (alchemy, for instance) have intensively tackled this issue, but scientific development in the Western world after the 'enlightenment' has excluded approaches in this direction. Like no other scientific approach of the last couple hundred years, 'systems thinking' clearly poses the relation of parts to a whole in the centre of reflection. Moving from technical to social aspects, the limits of the application of this approach become more and more visible. I am convinced we will be forced to revise our scientific foundation for systems development sooner or later. May be, 'anarchy thinking' can give some useful hints.

5. Anarchy in Systems development

In systems development, we use systems thinking to construct a mental universe for analysis, discussion, and further development activities. Being aware that the issues mentioned above are not settled at all, I try to give some examples how ‘anarchy thinking’ could possibly contribute to systems development projects:

Before starting a project ‘anarchy thinking’ calls for reflecting about the relevant social subjects to be involved. Besides individuals, collective actors must be considered as social subjects well (following Touraine 1984, collective actors have coordinated capability for action, explicit interaction with other actors, as well as a common perspective and goal-orientation). No given structure (e.g., an organizational chart) is to be accepted a priori. Any actor analysis should assume the absence of power relations until the project experience forces to revise these assumptions. A combination with ‘system thinking’ may be possible by creating actor models showing the relevant social subjects in the situation of concern.

While ‘system thinking’ in the context of software practice leads to the construction and implementation of algorithms, ‘anarchy thinking’ calls for social interaction and the freedom of decision about systems development issues. This interaction could ideally be a discourse, i.e. a ‘form of communication by reasoning to put problematic claims on the agenda and evaluate their justification’ (Habermas 1981). Making use of such a dualistic approach development methods could lead the actors involved choosing a path oscillating between discourse and algorithm implementation.

‘Anarchy thinking’ might also help actors involved in applying development methods, e.g.: (1) Control yourself as a method user but not the others! Gain knowledge and experience, evaluate and initiate discourse. (2) Don’t let ‘the method’ (or anyone else) control you! Question perspective and interest, and follow the method only when forced by authority.

Of course, these are only preliminary ideas, an input for discussion. It is an effort to overcome some limitations of thinking dealing with systems development and to scientifically acknowledge not only the structural, but also the anarchy aspects of real world projects. I am looking forward to these ideas being challenged during the Dagstuhl seminar, especially by those participants who favour dualistic approaches themselves.

References

Andersen, N.E., Kensing, F., Lundin, J., Mathiassen, L., Munk-Madsen, A., Rasbech, M., Sorgaard, P. (1990) *Professional systems development. Experience, ideas and action.* Prentice Hall, London

Checkland, P. (1981) *Systems Thinking, Systems Practice.* Wiley, Chichester (UK)

Feyerabend, P. (1975) *Against Method. Outline of an anarchistic theory of knowledge.* German edition: *Wider den Methodenzwang. Skizze einer anarchistischen Erkenntnistheorie.* Suhrkamp, Frankfurt/M., 1976

Habermas, J. (1981) *Theorie des kommunikativen Handelns.* Suhrkamp, Frankfurt/M.

Klischewski, R. (1996) *Anarchie – ein Leitbild für die Informatik.* P. Lang, Frankfurt/M.

Touraine, A. (1984) *Le retour de l‘acteur.* Fayard, Paris

Activity Analysis and Development through Information Systems Development

Mikko Korpela

In this position paper we propose theses – without proofs – that touch some fundamental issues of Information Systems (IS) and Information Systems Development (ISD): *What is the purpose of ISs and ISD? What are ISs? How can ISD be studied?*

IS practice and science is about *information technology in use in collective work settings*. IS as a science is thus a social science, closely related with Software Engineering in the way Architecture is related with Construction Engineering. The fundamental frame of reference of IS science and practice should be the 'collective work setting', not IS per se because the latter is a subordinated category.

'Collective work settings' are studied by Activity Theory. The frame of reference for IS in an Activity-Theoretical view – the unit of analysis – is a *collective work activity*. An activity comprises *a number of people working on something shared in an organized way – not necessarily at the same time and place – to produce a joint outcome*. The elements and relations making up the structure of an activity can be analyzed through a schematic model (Korpela 1999, Figure 1). Information technology can be used in various roles within activities. It is the activity which has a systemic nature, not necessarily the "information system" embedded in it.

Regarding what is usually tried to capture in the term 'information system', we speak about *the use of information technology (manual or computer-based) in a collective work activity, either as means of work or as means of coordination and communication.*

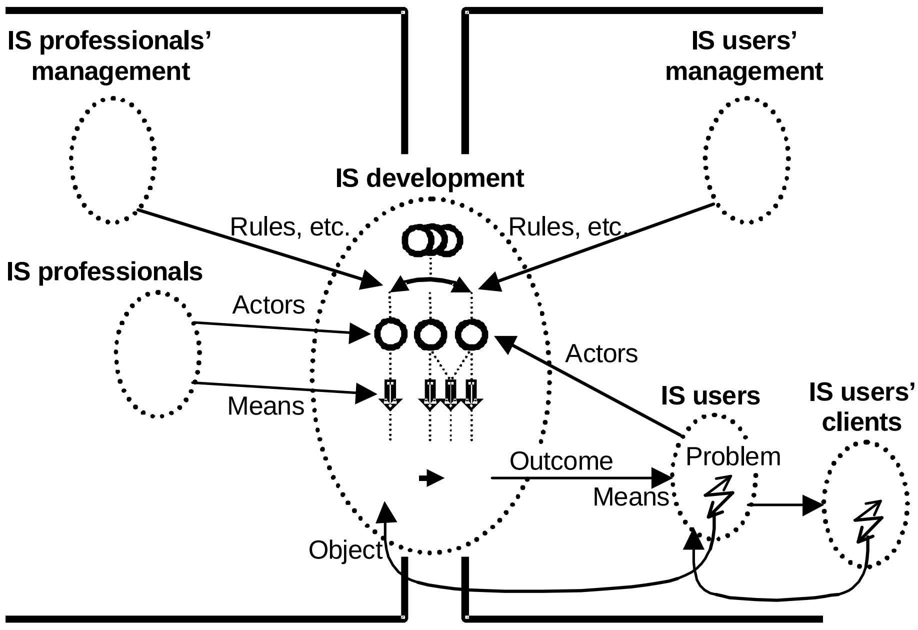
Correspondingly, we define that *information systems development is the process by which some collective work activity is facilitated by new information-technological means through analysis, design, implementation, introduction and sustained support.*

In ISD, the basic task is to analyze the work activity to be facilitated by the would-be IS, and to figure out how "misfits", "tensions" or "defects" within it can be soothed by information-technological means. Thus *work development*, or "activity development", is the purpose of IS development.

The most immediate *contexts* to be studied in ISD are the activity networks (Korpela 1999, Figure 2), organizations (Korpela 1999, Figure 3) and financial transactions surrounding the "host activity" of the IS.

ISD can be seen as a *boundary-crossing temporary activity* at the border of two departments, companies or other organizations (figure below).

IS research facilitates IS practice by theories, methods and skills (means). In order to produce appropriate facilitation addressing the most important needs of the practitioners, IS researchers must engage in a cooperative setting with IS practitioners to jointly analyze the latter's work activity (ISD) – in the same way as IS practitioners must engage in a cooperative relation with the IS users in order to produce appropriate facilitation to their clients.



The composition of IS development activity.

Acknowledgements

This paper is produced in the INDEHELA-Methods project funded by the Academy of Finland, and based on joint research work with Anja Mursu, Abimbola Soriyan, and Karen Olufokunbi.

Further reading

Activity analysis and development. Web site. http://www.uku.fi/atkk/actad/

Korpela M, 1999. Activity analysis and development in a nutshell. Handout. http://www.uku.fi/atkk/actad/nutshell.html

Korpela M, Mursu A, Soriyan HA and Olufokunbi KC, (submitted). Information Systems Development as an Activity. Submitted for refereeing to the *Computer-Supported Cooperative Work*, Special issue on Activity Theory and Design. http://www.uku.fi/atkk/actad/actad-cscw.doc

Korpela M, Soriyan HA and Olufokunbi KC, (forthcoming). Activity analysis as a method for information systems development: General introduction and experiments from Nigeria and Finland. In: Bødker S, Bertelsen O, eds., *Information Technology in Human Activity*. http://www.uku.fi/atkk/actad/is-actad.doc

---

Management and Work Practices in Software

Philip Kraft

In most discussions of software management practices there are typically four defining assumptions:

1. The management of software work is about the administration of specific work practices.
2. Work practices are about the divisions of labor in specific sites, that is, about who does what in relation to colleagues in the same or linked physical spaces.
3. There are "best practices" in the division of labor and management of software work.

4. The best practices/methods/systems which pour out of a more or less well-defined ribbon of intellectual geography from Silicon Valley to Munich are models for software producers in the rest of world. [I could have just as easily said from Seattle to Stockholm or from Menlo Park to Malmö. Please note, however, that the borders of the land of best practices do not include, for example, Bangalore or Panang. Or Tokyo].

The first three assumptions are often explicit and folded into concrete or theoretical studies of work practices in software production. The fourth assumption is typically less explicit but I believe just as deeply held. Together they reflect a preoccupation with software work as a problem of technical or organizational management: how to inspire or coerce software workers to most efficiently produce good software and good systems in good time. Typically the solution to this problem is to construct a collection of artifacts, that is, managerial strategies, which can be applied as needed. Although elusive and constantly changing, these best practices artifacts can be uncovered, systematized, and replicated. They can then be applied not just in the site where they were developed, but everywhere. Tools, methods, systems, and organizational practices developed in Seattle or Pittsburgh or Cambridge or Oslo will eventually become the global norm. The process may happen unevenly, but it will happen inevitably and it will happen, sooner or later, everywhere.

Taken individually and collectively, I think all four assumptions are at best incomplete. By way of illustration, let's look at an issue which currently occupies students of software work practices: so-called offshore programming. To quote one recent Danish paper, offshore programming is "the practice of sending software development 'offshore,' to India or other low wage areas." The other low wage area mentioned in this example is Eastern Europe. Unstated but assumed is that offshore work is commissioned from western Europe and North America. But these two givens – low wages and Asian locations – present certain analytical problems. For example, if Americans or Danes call the making of software done in, say, Bangalore, "offshore programming," what do people in Bangalore call the making of software done in the US or Denmark? Further, the UK, Scandinavia, and Germany are all sites of large US investments in software production. None of these can be thought of as low wage countries. For US firms these are every bit as offshore as India or the Philippines or Hungary or Russia.

Let us agree that software wage costs are considerably less in India than in the US. They are probably as cheap in Russia, not quite as low in Hungary, and less cheap but still a bargain in another favorite offshore site of US firms, Ireland. When we move to the UK, the differential with the US is substantially reduced, and in Germany and Scandinavia direct wage costs are more or less the same or even higher. But here then is another puzzle: if it is simply a matter of cheap wages, why is there any programming work of any kind left in, say, Germany, when Siemens and Nixdorf and Deutsche Telekom, as well as IBM and Hewlett-Packard, employ cheap Hungarians and still cheaper Russians?

My colleague Richard Sharpe's typology of primary, secondary, and tertiary software turns out to be extremely useful in making sense of this apparent anomaly. In brief,

Primary Software is produced in centralized locations and sold as commodities to users for further exploitation: operating systems, applications packages, development tools, etc.

Secondary Software is built using these primary software products. These include but are not limited to customized applications packages, e.g., billing systems, on line processing systems, manufacturing control systems, etc.

Tertiary Software is made by individual end users for their own individual or small scale application, e.g., scripts, macros, applications software settings, etc.

Sharpe's typology is useful because divides software work along a continuum of social and political criteria in addition to the usual categories of technical complexity or cost. Using it we can formulate some fundamental questions:

- Why do the design and development of systems software and "killer apps" take place chiefly in the US (with apologies to Baan, SAP, etc....)?
- Why do some – but not all – Northern European countries specialize in niche products or adapting US designed software to local markets?
- Why do Asian and East European software workers seem to specialize chiefly in searching for Y2K bugs, maintaining COBOL programs for North American and European banks and insurance companies, or modifying database software produced in western Europe or the US?

Let's rephrase these questions: if software is the ultimate "informational" product, to borrow a terms from Manuel Castells, why aren't all types of software, from operating systems to hypertext helpfiles, produced everywhere? More to the point, why is there an international division of labor that restricts systems level R&D to North American and to a few, very few, European locations, limits specialized applications development to parts, but not all, of Europe, maintenance to Bangalore and Manila, and ***provides essentially no software work of any kind in most of the rest of the world***? In other words, if the virtual village, the networked economy, and the global economy are truly upon us, why aren't Ethiopians and Bolivians making new operating systems? For that matter, why aren't Italians or Japanese?

Obviously, wage costs are important in any specific decision to produce a particular piece of software in a particular place. But this begs the question. If wages are cheap in Bangalore compared to San Jose or Westchester or Geneva, why not move design and product development, as well as the routine stuff, to Bangalore? The programmers who work at the various Technology Parks there are smart, they work hard, they have all sorts of microwave telecommunications and computing hardware, and there are lots of them (which is why they are cheaper than Americans or Germans or Danes).

Using Sharpe's typology there are several possible ways to pursue such questions without resorting to the problematic explanation of wages. Here are a few Richard and I are examining now:

1. Software work practices differ because software work is not universally distributed; it is selectively distributed. How is the organization of specific software work sites affected by this distribution process?
2. What considerations – apart from obvious incentives like wages – prompt firms to divide software labor geographically as well as intellectually? Why must OS software or the design or other integrated systems be restricted to certain geographic areas while the maintenance of shrink-wrapped consumer software can be more broadly – although not globally – dispersed?
3. Workforces in these selectively distributed sites have their own histories and contexts. Only some of these are technical; the others are political. What are the connections to state employment and training policies? subsidized wage costs via social insurance? colonial histories? For example, the oft-cited competitive advantage of Indian and Irish programmers – they are English speakers – is due to their colonial histories; what else has this colonial heritage meant? Taiwan and South Korea, Eastern Europe, and Brazil are interesting for analogous reasons.
4. Different histories and contexts mean that relations ***between*** worksites as well as ***within*** worksites are likely to differ. What are the similarities and differences in the ways owners and managers in the same international firm assemble, train, control, and pay software workers in different places, e.g., what administrative and control practices do Hewlett-

Packard managers use in Silicon Valley versus in Silicon Glen? In Japan? In Panang? How different is Motorola in the US and in India? Fujitsu in Japan and the UK?

These are not the only questions a broader conception of software work practice allows. They may not even be the most important ones. They do, however, permit us explore the division of software labor and the social relations of software workplaces without reducing either to a series of abstracted and isolated "practices" or "tools."

Use Oriented Modelling

Olle Lindeberg

My primary research interested is in design and modelling of software. I want to study how this is done in practice not only by software developers but also by users. This demands software projects that use some sort of participatory design.

My interest comes from my own experiences. After my undergraduate studies (a master in physics engineering with all the computer science I could get) I worked several years as consultant – primarily working with embedded systems and real-time programming. Even if I am now more interested in social embedded systems this experience told me that the user (or client) view of the software was often quite different from my own as software developer. My studies in computer science gave me no help in bridging this gap and it is only now that I have found literature making it possible to express these issues

In social embedded systems the design of the users conceptual model of the system is a key issue to make a usable system. In software development a key issue is what Peter Naur called the theory building. Theory here means the understanding the software developers have of how the software works, both internally and in connection with the environment. I want to study the relationship between these different views of the software system.

My interest in modelling comes from my earlier work at Lund University where I did a Licentiate thesis about mathematical modelling of knowledge bases. In working with this I got an understanding how formal systems and formal modelling work. The research was focused on basic theoretical issues far from any user or software developer. During this work I got curious about how much of formal modelling is usable in practice and now want to study how users and software developers use models in practice.

The last half year I have been working with the group People, Computers and Work at the University of Karlskrona/Ronneby. This is an interdisciplinary group consisting of both computer science and work science. This was for me the first contact with the use of ethnomethodology methods to study work practice before starting to designing software. It also opens the possibilities to use the same methods to study the work of the software developers. The contact with researchers from other disciplines have changed my view of software development and open up possibilities to investigate questions that I been thinking about since my own work as a software developer.

My own experience as software developer tells me that what really occurs in software development is not at all what you learn when study software engineering or computer science at a University. It is therefor necessary to participate in real projects to make this research. The methods to be used in such research depends on the project and I am now in the process of starting cooperation with industry to start my empirical studies.

Social Thinking and Software Practice: One common platform -or two platforms?

Hans-Erik Nissen

Introduction: The expressed aim of the seminar

In the motivation to this seminar I found the question: "What is the relation between, on the one side, reflective and analytic abstractions of the social sciences and, on the other side, generative abstractions developed by designers in order to implement in and with a computer application?" The motivational text then declares a coherent understanding of this and similar questions as needed to provide suitable methods for software development.

The focus on abstractions can make a reader presume those stating the question believe - maybe unconsciously - in a neat separation of theory from practice. The declared aim of providing suitable methods of software development to me supports such a conjecture.

The motivational paper states as an aim of the conference: "… to promote a conversation … which might lead to establishing of a common platform. …" As the writers did not write "common platforms" I choose for my argument to interpret the aim as an aim for *one* common platform. The IFIP Working Group 8.1, Design and Evaluation of Information Systems, as early as 1987, in their manifesto for the so-called FRISCO task group, wrote:

"There is a growing concern within the IFIP WG 8.1 about the present situation, where too many fuzzy or ill-defined concepts are used in the information systems area. Scientific as well as practice-related communication is severely distorted and hampered, due to this fuzziness and due to the frequent situation that different communication partners associate different meanings with one and the same term. There is no commonly accepted conceptual reference and terminology, to be applied for defining or explaining existing or new concepts for information systems." (Falkenberg, et al., 1998, p.4)

The FRISCO task group published a "final" report late 1996. In this the group states:

"The real concern (the misunderstanding of what is involved in organisational communication) is still there - in spite of our studies - and one may fear that some of the problems are innate to the various interested parties. The roots lie in the past, for the information systems area has suffered from the fact that due to historic accidents, different facets of the area have been dealt with by different scientific disciplines or "cultures", in particular computer science and the social sciences…"

The FRISCO group itself underestimated these problems, in particular the existence of "hidden agendas" of the interested parties. …" (Falkenberg, et al., 1998, p.4)

The report starts with a broad conceptual outlook. Still, the group devoted most of its efforts to produce an extensive catalogue of formal definitions of concepts. Two of the group members have, versus the end of the report, criticised the dominant position given to formalism. Very simplified, I here spot a tendency giving scientific status to the report by keeping close to the ideal of the program of unified science advocated by the members of the Vienna Circle in the 1920s - 1930s. I feel concerned that this seminar, and what ensues from it, unconsciously, by aiming for creating *one* platform, could fall back to ideas fetched from the ideals of unified science. The Encyclopædia Britannica briefly presents the Vienna Circle:

"German WIENER KREIS, a group of philosophers, scientists, and mathematicians formed in the 1920s that met regularly in Vienna to investigate scientific language and scientific methodology. The philosophical movement associated with the Circle has been called variously logical positivism, logical empiricism, scientific empiricism, neopositivism, and the unity of science movement. The work of its members, although not unanimous in the treatment of many issues, was distinguished, first, by its attention to the form of scientific theories, in the belief that the logical structure of any particular scientific theory could be specified quite apart from its content. Second, they formulated a verifiability principle or criterion of meaning, a claim that the meaningfulness of a proposition is grounded in experience and observation. For this reason, the statements of ethics, metaphysics, religion, and aesthetics were held to be assertorically meaningless. Third, and as a result of the two other points, a doctrine of unified science was espoused. Thus, no fundamental differences were seen to exist between the physical and the biological sciences or between the natural and the social sciences." (Britannica CD. Version 97)

A more extensive discussion of the Vienna Circle and unified science can be found in Radnitzky (1970, Vol. I, pp. 72 - 92). This brings me to a discussion of some schools of meta-science.

Two main schools of meta-science

Even researchers within social sciences have undertaken their studies from rather different meta-scientific perspectives. Here it suffices to point to the classification of Theories of Organising presented by Burell and Morgan (1979).

In my discussions of meta-science I will follow Radnitzky (1970). He distinguishes Anglo-Saxon traditions, which he calls Logical-Empirical (LE) and Continental traditions, which he calls Hermeneutic-Dialectic (HD).

The LE traditions have produced "…precise adequacy criteria for the appraisal ex post of certain finished products of science judged with respect to formal aspects, … What we must be aware of, …, is the fallacy of totalization …" (Radnitzky , 1970, Vol. I, p. 69) Radnitzky then goes on to argue: "… If the metascientist is to assist the scientist, criteria governing the *production* of knowledge and of techniques, and an adequate theory of *growth* of scientific knowledge as a whole are far *more important* than criteria for appraising finished products. …" (ibid.)

My concern with attempts to establish *one* common platform I base on a risk I perceive they will end by a platform building only on the meta-scientific positions of LE. In the context of social sciences this would entail communication only with social scientists belonging to schools of functionalist sociology. (Cf. Burell and Morgan, 1979, chapters 4 and 5.) The social scientists participating in the seminar did not belong to these schools, as far as I could understand. Nor do I believe the software science participants at the seminar based their work only on LE positions. However, within the set of texts even reaching software science readers outside of the participants a reminder of the risk of establishing *one* common platform I deem warranted. My reason for this conjecture is the fact that most of us, on the software science side, have a solid introduction and training in one LE tradition or another taking its positions as granted. Empirically I ground my conjecture in the above-mentioned experience of the FRISCO group.

In order to choose meta-scientific positions consciously I presume it helps to make the differences between the positions of the two main schools explicit. The following features by and large characterise LE traditions:

- They presume one observer independent "reality".
- They predominantly apply correspondence theories of truth.
- The historical context of data collected does not play any role.
- They only accept science-immanent values as guiding research,
- which makes theory separable from practice.
- A tacitly presumed technical research interest.
- Stress on extensional language and denotational features of language.

The HD traditions have mainly evolved together with the so-called cultural sciences. Studies by social scientists exist both in LE and HD traditions. (Cf. Burell and Morgan, 1979) The following features by and large characterise HD traditions:

- They acknowledge observer related "realities".
- They apply coherence theories of truth.
- The historical context of data collected plays a role.
- They also acknowledge values outside of science-immanent ones as guiding research, which makes theory and practice non-separable.
- They acknowledge also other research interests than technical ones, e.g.. mediating tradition, and emancipatory ones.
- Stress on intentional language and connotational features of language.

Research interests can only become discussed within a meta-scientific frame comprising both these streams of research traditions. Let me also state that, as argued by Apel (1967, pp. 56-57), methods of inquiry only fetched from HD traditions will not suffice in social science studies. As his reason Apel points to the fact that as humans we do not fully understand each other and ourselves. To illustrate his point Apel starts from the *development of knowledge and self-understanding* in an ideal-typical psychoanalytic situation. From Radnitzky (1970)I have fetched the following explanation of Apel's idea:

"We shall consider a psychoanalytic situation where the analysand wishes to carry out a personal analysis in cooperation with the analyst to learn more about his operant motivation and also to increase his psychological possibilities of carrying out his conscious intentions. The *global or final interest* motivating the whole enterprise, on the part of the analysand is … the *hermeneutic one combined with the emancipatory one*. Having agreed upon the global program, *resource-gathering* may begin. The *dialogue* between analysand and analyst supplies material … At this stage, among the resources of the analyst, the prenotions which he has qua member of the "same" speech community as the analysand are of particular importance. … it is required that the analysand and the analyst speak the "same language" - to ensure that they have sufficient shared common life experience … (In the case of ethnographic studies the inquirer must first acquire familiarity with a form of life not his own. …) At this *hermeneutic* level or phase the analyst may achieve *Vertstehen* directly, i.e., it is attainable by the good-reason-assay utilizing the aforesaid prenotions. …

As long as attempts to give good-reason-assays are successful …, the dialogue is continued." Radnitzky (1970, Vol. II, pp 43-44)

Sometimes it occurs that the analyst can no longer make sense of what the analysand says or does. They reach a crisis in their co-understanding. The psychoanalytic analyst then can draw upon another resource, not shared with the analysand, psychoanalytic theory. This theory both offers systematised experience gathered by psychoanalysts and a special sublanguage, which also comprises a causal language. In that language the analyst can think and speak of "motives" as causal factors. The analyst, drawing on psychoanalytic theory, may become able to explain the speech and other behaviour of the analysand. When he does, however, he does not use such an explanation to manipulate the analysand. He goes back to the dialogue situation and presents his theory based insights in a way that makes them reasonably understandable within the life history of the analysand. In other words the analyst returns to the agreed upon global program of the consultation, i.e. for the analysand to learn more about his operant motivation and also to increase his psychological possibilities of carrying out his conscious intentions.

Such turns to use what parallels natural science methods of inquiry - in service of improved understanding - Apel and Radnitzky (1970) call quasi-naturalistic turns. They vastly increase the resources of the psychoanalyst or of any other professional specialist - even when attempting to contribute to a research interest of mediating traditions or an emancipatory one.

The dividing line between LE and HD does not follow in which tradition a method of inquiry originally became developed. Instead it goes between using new knowledge exclusively for some people to manipulate and control others or in the first place to let the human 'objects' of a study also become its 'subjects' who can incorporate its findings into their own language, self-understanding, and future action. This means: Questions about who could act more adequately after acquiring some new knowledge should precede questions about methods of investigation to use.

Research interests and who could act more adequately on new knowledge?

If we as researchers want to take advantage of what we can learn from metascientists, we should start our research programs by asking ourselves two types of questions. (i) Which kind of research interest(s) do we intend our program to serve? (ii) Who could act more adequately based on the possible findings of our research program? These questions, probably, will look like sheer nonsense to researchers strictly adhering to LE traditions. To make sense they presume a meta-scientific position broad enough to cover both LE and HD traditions. Moreover, they have to become interpreted presuming theories and their applications in practice cannot be strictly separated.

Even accepting these premises it might seem unfair to ask a researcher to take on responsibility for how her findings will become used and applied. These questions, of course, also raise important issues of research ethics, treated a lot lately. Still, I choose not to discuss research ethics further in this paper. I raise the two questions above as fundamental at an early phase of planning a research program.

Although working with the intention to pursue a tradition mediating (hermeneutic) or emancipatory research interest the findings of a research program might become used/ misused by others with a technical research interest. However, deciding on strictly adhering to LE traditions in order to work "scientifically", only to pose "researchable questions", etc. will hardly lead to findings, which can serve hermeneutic or emancipatory research interests. Raising the second of the two questions above at an early stage of planning a research program or project entails the following kinds of considerations. Will those intended to become enabled to act more adequately based on research findings be able to afford the instruments used? If so, still, will they have the competence to do so or would that demand some learning on their part? These and similar questions researchers better raise at the research planning stage.

Radnitzky (1970, Vol. II, pp. 5-14) partly drawing on Habermas, but also adding some research interests himself, distinguishes the following five research interests:

- Technical
- Mediating tradition(s) (Hermeneutic)
- Emancipatory
- Improving the world picture
- Improving the reflection about existential themes

Much, if not most research, on software development intends to serve a technical research interest. Research on implementation and use of new IT-artefacts may serve a technical research interest of managers to control their subordinates in order to get more value for money put in their IT investments. Alternatively such research could intend to serve an emancipatory research interest to let IT-artefact users gain more influence over their work situations. Most probably different research methods have to become chosen for the two different guiding research interests.

All the research interests mentioned can legitimately become followed. Making their research interests explicit, already when planning a research program, helps the group of researchers perceive limitations to the scope of their findings. It also helps in framing research questions and in choosing research methods adequate to the intended research interest.

Answers to the second kind of questions above also will influence choices of research methods. Particularly, resources available and ways of action culturally open to those intended to act differently have to become considered.

Offering a smorgasbord of different research approaches without criteria for their selection and combination in relation to the intended aim of a research program to me seems a big undertaking with limited promise of success.

Establishing one platform or an ongoing dialectic between two platforms?

Let me start with a conjecture: This seminar, and what may ensue, stands a better chance of lasting contributions by aiming at an ongoing dialectic between two platforms for research on software development in social contexts than by aiming at one common platform.

What arguments can I offer for this conjecture of mine?

In the introduction I told the experience of a similar attempt to establish one common, conceptual platform for information systems research. In my next section I briefly sketched two rather different schools of meta-science. In the following section I introduced five different research interests. I also introduced the idea that if research findings become applied at all, *somebody* will apply them. The broad range of research interests and the different value sets of those going to apply research findings seem to stand a better chance of becoming covered by two different platforms than by *one* common platform.

What do I suggest could characterise the two different research platforms? As one platform I tentatively suggest some pragmatist variety of LE traditions supplemented with insights and methods from hermeneutics used in explicitly treating how self-understanding can become improved in their communities of researchers. (Cf. Apel, 1967) As the other platform I tentatively suggest some variety of HD traditions supplemented with what Apel has called quasi-naturalistic phases of inquiry and explanation. (Radnitzky, Vol. II, pp. 44-47) On the one hand none of the two platforms coincides completely with one of the two main schools of meta-science. On the other hand I have in both cases only suggested limited parts from the other research tradition to become included.

Bateson (1967) has argued the need of two different types of inquiry one for studies of nonliving systems and another for living systems. Currently I choose to talk about a distinction between studies of non-autonomous and of autonomous systems. The very distinction made seems to give some support to my conjecture of introducing two platforms.

The first mentioned platform best supports research programs with a predominantly technical research interest. To these many research programs in software development, but also in organisational development (following what Burell and Morgan (1979) call functionalist traditions) will belong. This platform will also best support the research interest of improving the world picture with respect to cosmology as opposed to with respect to anthropology.

The second research platform best supports research programs intended to serve non-technical research interests. To these research programs on software use, and its effects as well as research programs studying software implementation in organisational and societal contexts will belong.

None of the two platforms sketched should become looked upon as static. Both need further clarification and development. Such clarification and development would profit from becoming undertaken as an ongoing dialectic between advocates for the two different platforms.

What next?

In this paper I have interpreted the aim of this seminar as starting to establish *one* platform for social thinking and software practice. I gave some historical evidence that such a program might turn out as infeasible in historical communities of researchers. Still worse I saw it entail a risk of ending as a modern revival of a school of meta-science closely related to the unified science movement of the 1920s and 1930s. As an alternative aim for the seminar I proposed establishing two different research platforms developing through an ever ongoing dialectic between the advocates of one or the other of these.

Now I need comments from my peers at this seminar and other readers. Have I misinterpreted the aim of the seminar? Does someone share my diagnosis the seminar addressed meta-scientific questions, but did not frame them as such? Does someone share my view that in the context of studies and methods of software development a need for strict formalism exists on the one hand but should not become the only platform for all research? So next I look forward to a chance for debating these and similar questions with some of you my readers.

ACKNOWLEDGEMENT

In the revised version of this paper I have taken advantage of advice given by Pekka Reijonen at the University of Turku, Finland. I gratefully acknowledge his advice and suggestions. I also like to thank participants at the Dagstuhl seminar for questions and discussions about my concern. The way I have used the viewpoints I received, of course, remains my responsibility.

REFERENCES

Apel, K.-O., D. Reidel (1967) Philosophy of Language and the Geisteswissenschaften Dordrecht, The Netherlands, first published in German in 1965.

Burell, G., Morgan, G. (1979) Sociological Paradigms and Organisational Analysis Elements of the Sociology of Corporate Life Heinemann, London

Bateson, G. (1979) Mind and Nature A necessary unity Fontana Paperbacks, London, 1985

Britannica CD. Version 97, Encyclopædia Britannica, Inc., 1997

Falkenberg, E. D., Hesse, W, Lindgreen, P., Nilsson, B. E., Oei, J. L. H., Rolland, C., Stamper, R. K., Van Assche, F. J. M., Verrijn-Stuart, A. A., Voss, (1998) A Framework of Information Systems Concepts The Frisco Report (Web edition) IFIP WG 8.1, originally completed in Dec. 1996

K. Radnitzky, G. (1970) Contemporary schools of metascience, Vol. I: Anglo-Saxon schools of metascience, Vol. II: Continental schools of metascience, Akademiförlaget, Göteborg, Second revised edition in one volume

Software process improvement: controlling developers, managers or users?

Jacob Nørbjerg

1. Introduction

A common view of software development organisations is that they are here to serve the "users"[1], by developing the (high quality) computer based information systems the users need and as cheaply as possible. The challenge is to identify that system and decide how to develop it. At this Dagstuhl we are going to discuss some of the possible answers to that challenge.

But software work is work too, and development organisations are organisations like other organisations. Software organisations have their own set of work practices, technologies, structures, management practices, etc., and developers and managers in software organisations pursue their own goals and interests. The interaction between development and user organisations is therefore marred with inter- and intra-organisational conflict and politics like any other organisational or work relation. Deciding what the needs are, what system will meet those needs, how the system shall be developed, who shall control the process etc. etc. are not only issues to be solved through debate, but battlegrounds for organisational politics and power games within, as well as between, both the user and the developer organisations (Knights and Murray 1994).

This implies that changing software development practice is not only a matter of better "techniques", but involves fundamental issues of organisational structure, management practices, and organisational power and control (cf. Franz and Robey 1984, Grudin 1991a, Grudin 1991b, Knights and Murray 1994).

In this paper I will (briefly) discuss how the latest trend in the management of software development: software process improvement (SPI) may affect user-developer relations. At the outset, SPI concerns the "internal workings" of software organisations, but it may also be interpreted as one way to give the developer organisation more control over the development process.

The discussion is based on my involvement in an SPI research project over the last 2 1/2 years (cf. Iversen et. al. 1998, DELTA 1998, Mathiassen and Johansen 1998) but is mainly speculative (not based on systematically collected data, but on observations and analogy to other research).

2. Get the process under control!

In contrast to previous (failed) attempts to control (and improve) software development through "isolated" tools and techniques; e.g. structured methods, CASE-tools, project management, testing techniques etc., is SPI about changing the whole of the software organisation. The fundamental view is, that improvements to software development[2] require co-ordinated and carefully planned changes to the whole process: project and people management, development techniques and tools, organisational structure, quality control and improvement, productivity and quality measurements, etc.

To prioritise and co-ordinate this change effort, SPI applies the idea of *software process maturity*. Software process maturity can be interpreted as a "measure" of an organisation's ability to control its software development. The idea of maturity provides not only the measuring

[1] The usual reservations regarding the meaning of "user" apply.

[2] "Improvements" means better ability to meet cost and quality targets, and – ultimately – to reduce cost and improve quality.

scale, but also the map to follow in the improvement process. Basically, an immature software organisation should address fundamental project management practices first, then proceed to implementation of organisation-wide development standards, and finally take on systematic process and product measurements and improvements to productivity and quality[3].

The debate about SPI in the software engineering community has centred around the implications for creativity and flexibility in software development. That is, whether the increasingly defined and documented processes associated with higher maturity will inhibit developer creativity and make the development organisation too inflexible to meet today's rapidly changing market requirements (cf. Bollinger and McGowan 1991, Kohoutek 1996)

Another interesting discussion concerns of course the management and control of developers. It is easy to draw up scenarios of increased levels of management control as maturity grows. The picture is more complex, however, since high maturity puts pressures and demands on management too. Management in the high-maturity organisation commits itself to a systematic planning process, based on past statistics and the professional judgement of software professionals, where all changes to project conditions (i.e. reallocation of developers, requirements changes etc.) result in new plans. This gives, at least in principle and in the rhetoric of SPI, the developers some leverage in the day-to-day battle with managers over time and other resources.

In the rest of the paper, I will, however, focus on (possible) implications for the relations between the user and developer organisations.

2.1 Controlling users?

I will address two issues: the control of user-developer interactions, and the use of statistics in negotiations with the users.

Control interaction

"The biggest problem for the mature development organisation is the immature user

This remark[4] from the instructor at an SPI tutorial illustrates a possible outcome of increased software process maturity: The mismatch between the user organisation's sloppy practices and the clearly defined and documented procedures in the (mature) software organisation.

Among the well-defined procedures in the mature software organisation are those that regulate relations between users and developers, as for example, change management, planning (and re-planning) and estimation procedures. This means that interactions between users and developers, particularly interactions that may result in changes to the process or the product, are formalised and channelled through specified routes (customer relations offices, change control boards etc.). The mature software organisation will no longer allow the kind of ad hoc interaction and change typical of immature software processes and (presumably) more or less taken for granted by (immature) user organisation.

This, first of all, may give the development organisation more firm control over the process and the product, but it will also reduce the possibility to take advantage of the learning and insight that develops continuously in the cause of development projects (cf. Curtis et. al. 1988). Furthermore, the reductions in informal developer-user interactions will shut developers off from an important source of information about the application domain and the users' needs (cf. Beirne et. al. 1998, Curtis et. al. 1988, Nørbjerg 1995).

[3] For definitions of maturity and approaches to the assessment of maturity see (Humphrey 1989, Kuvaja et. al. 1994, Paulk et. al. 1993, Rout 1995)

[4] This and the subsequent remark are "quoted" from memory. The quotes represent the "meaning" of the statements, not the exact wording.

Controlling the numbers

Metrics and measurements play an important role in SPI: the more mature the organisation, the more data about the process are collected and used to plan and monitor projects, as well as evaluate and improve productivity and quality of process and product.

The numbers are collected and used primarily for internal managerial purposes of course, but having (and controlling) statistics about past performance are important in negotiations with users too: Having the numbers to document past performance is an important asset in negotiations with the user organisation. Or, as a instructor at an SPI tutorial put it:

"If you have the metrics in place, you can document to the customer, that developing this system, to that level of quality, will cost so much and take so much time."

The problem is of course, that the interpretation of any measurement of software process performance is highly contextual (Sommerville 1996) and subject to organisational politics (Iversen 1999, Kohoutek 1996). Nevertheless, numbers are imbued with an air of authority and rationality, and since the development organisation "controls the numbers" it also controls which numbers to present and their "proper" interpretation[5].

3. Relation to the Dagstuhl theme

Software development is a human activity, that takes place within specific organisational circumstances and conditions. These circumstances are as much part of "software practices", as any isolated "method" or "approach" applied by the developers.

In this paper I have discussed how the development organisations can use SPI to improve their control over the development process at the expense of the users and the user organisation. The implications for the implementation and use of "social theory" in software development are unclear (as yet) – but they are worth considering.

References

Beirne, H., Ramsay and H. Panteli, A. (1998): Developments in Computing Work: Control and Contradiction in the Software Labour Process, in, Thompson, P. and Warhurst, C. (eds) *Workplaces of the Future*, Macmillan, London, p. 142 - 162.

Bollinger, T. B. and McGowan, C. (1991): A Critical Look at Software Capability Evaluations, *IEEE Software*, 8(4), 25-41.

Curtis, B., Krasner, H and Iscoe, N. (1988*)* A Field Study of the Software Design Process for Large Systems, *Communications of The ACM,* 31 (11), p. 1268-1287.

DELTA(1998): *Danske erfaringer med forbedring af software processen* (*Danish experiences with software process improvement* - in Danish), Technical Report, D-262, DELTA.

Franz, C. R. and Robey, D. (1984): An Investigation of User-Led System Design: Rational and Political Perspectives, *Communications of the ACM,* 27 (12), p. 1202 - 1207.

Grudin, J. (1991a): Interactive Systems: Bridging the Gaps Between Developers and Users, *IEEE Computer* , 24, p. 59 -69.

Grudin, J. (1991b): Systematic Sources of Suboptimal Interface Design in Large Product Development Organizations, *Human-Computer Interaction*, 6, p. 147 - 196.

Humphrey, W.S. (1989): *Managing the Software Process*, Addison-Wesley, Pittsburgh, Pennsylvania.

Iversen, J., Nielsen, P. A. and Nørbjerg, J. (1998): Problem Diagnosis in Software Process Improvement, in, Larsen, T. J., Levine, L., DeGross, J.(eds.): *Proceedings of IFIP 8.2 and 8.6 Working Conference*, IFIP, p. 111 - 130. (http://www.bi.no/dep2/infomgt/wg82-86/proceedings/index.htm)

Iversen , J. (1999): Establishing SPI Effect Measurements, PROFES 99.

[5] Tales of *not* publishing numbers that are unfavourable to the development organisation can be told too.

Knights, D. and Murray, F. (1994): *Managers Divided. Organisation Politics and Information Technology Management*, Wiley.

Kohoutek, H. J. (1996) Reflections on the capability and maturity models of engineering processes, *Quality and Reliability Engineering International*, 12(3), 147-155

Kuvaja, P., Similä, J., Krzanik, L., Bicego, A., Saukkonen, S., and Koch, G. (1994): *Software Process Assessment and Improvement - The BOOTSTRAP Approach*, Blackwell.

Mathiassen, L. and Johansen, J. (1998): Lessons learned in a National SPI Effort, *EuroSPI'98*.

Nørbjerg, J. (1995): How developers learn. Skill distribution and skill building in systems development, in, Dahlbom, B. et. al., *Proceedings of IRIS18*, Gothenburg Studies in Informatics 7/1995, p. 609 - 624.

Paulk, M.C., Curtis, B., Chrissis, M.B., and Weber, C.V. (1993): *Capability Maturity Model for Software, Version 1.1*, 93-TR-024. Software Engineering Institute, Pittsburgh, Pennsylvania.

Rout, T.P. (1995). SPICE: A Framework for Software Process Assessment, *Software Process - Improvement and Practice*, 1 (Pilot Issue), 57-66.

Sommerville, I. (1996): *Software Engineering*, Addison-Wesley.

Unit of Analysis: Frame of Work

Markku I. Nurminen

This position paper is prepared to the session on *Frameworks for understanding social aspects related to software development.* It consists of two parts. In the first part I have made some interpretative parsing of the theme of this session. In doing this I have tried to explain my understanding of (position relative to) the title. The second part contains some sketches of a possible framework which could integrate different perspectives. They are based on the belief that the "reality" is to a great extent the same for all observers. Differences come from the use of different angles of observation, which often begin by selecting and excluding different qualities of the object of study. Thus, it could be possible to agree about a shared understanding of the basic unit(s) of analysis.

1. Parsing of the theme

I shall proceed from the end to the beginning since it seems to uncover the meaning in a logical order. I read the title in the role of a devil's advocate exaggerating all opportunities for misunderstanding. In doing so I hope that I can determine more accurate coordinates to my current position.

Development: intentional and/or emerging change of the object of study, justified by the desired outcome, not by the process itself.

Software: As a separate construct it does not make sense. The ontology of software should be reformulated to be compatible with a set of socially interpreted parts to be applied in (different) work practices.

Related to: The primary thing here seems to be the software development in a way which would allow some social aspects to be glued to it. In my mind there is no reason to accept this kind of autonomy of technology.

Aspects: Who invented the existence of such aspects. The formulation indicates that it has turned out that the "autonomous" software development (understood on its own premises) is not sufficient; therefore some social aspects have to be identified as auxiliary labels to be glued to the real thing, SD.

Social: This confirms some of my previous doubts. Software development is here understood as something non-social in its essence. The only way to relate to the social world is to list some aspects which are, of course, external.

Understanding: The social aspects seem to be findings of exploratory research and we have trouble in understanding them. Social aspects have somehow emerged while we have been observing the phenomenon SD and we now have to make sense of them, in retrospect.

Frameworks: The aspects should be arranged to a coherent composition and this picture should be surrounded by a frame before hanging the picture on the wall.

2. Towards a framework: In search for a unit of analysis

It is obvious that we would benefit if we could agree about a shared area of study and the basic units of analysis within this area. Then we would have entities which can carry attributes. Entities and attributes give us the possibility to say that two instances or classes are similar or different in respect of one or more attributes, because then we know that we are talking about the same thing. This is a great advantage in a scientific discourse, but it is also useful for talking about development: what is going to be changed and which attributes are subject to change.

I suggest that we try to find the basic unit(s) of analysis in the sphere of work practice rather than in software or software development, since work is more fundamental than the artefacts, even if the work often is mediated by artefacts. Within this domain, I try to formulate a few requirements or desired properties for such basic units.

The unit should give space to both external and internal sides of work in a balanced way. For example, business processes and outcome requirements should be supplemented with articulation and awareness. There are characteristics, such as knowledge and responsibility, which in itself bridge this dualism.

We need units of analysis which represent activity. These units should qualify for dealing with different levels of complexity of activity from simple work processes to large collective activities (scalability☺). The units should have concepts for collaboration in organisations with more and less specialised division of labour. The unit of analysis also carries its justification of existence and criteria of good performance consistent with this justification.

In my recent work I have tried to articulate some of the main characteristics of such a unit of analysis as a unit of activity. Activity theory seems to be useful, even if there are many open holes which have to be filled before the construct meets the requirements outlined above. In this seminar my intention is to evaluate other frameworks in order to find out the consistencies and inconsistencies in the use of such a unit of analysis.

Interaction Design:
Bridging Social Thinking and Software Construction

Horst Oberquelle

In place of an introduction

From:

Alan Cooper: The Inmates are Running the Asylum. Why High-Tech Products Drive us Crazy and How to Restore the Sanity. SAMS, Indianapolis, 1999, pages 15-16

"An Industry in Denial

We are a world awash in high-tech tools. Computers dominate the workplace and our homes, and vehicles are filling up with silicon-powered gadgets. All of these computerized devices are wildly sophisticated and powerful, but every one of them is dauntingly difficult and confusing to use.

The high-tech industry is in denial of a simple fact that every person with a cell phone or a word processor can clearly see: Our computerized tools are too hard to use. The technologists who create software and high-tech gadgets are satisfied with their efforts. The software engineers who create them have tried as hard as they can to make them easy to use and they have made some minor progress. They believe that their products are as easy to use as it is technically possible to make them. As engineers, their belief is in technology, and they have faith that only some new technology, like voice recognition or artificial intelligence, will improve the user's experience.

Ironically, the thing that will likely make the *least* improvement in the ease of use of software-based products is new technology. There is little difference *technically* between a complicated, confusing program and a simple, fun, and powerful product. The problem is one of culture, training, and attitude of the people who make them, more than it is one of chips and programming languages. We are deficient in our development process, not in our development tools.

The high-tech industry has inadvertently put programmers and engineers in charge, so their hard-to-use engineering culture dominates. Despite appearances, business executives are simply not the ones in control of the high-tech industry. It is the engineers who are running the show. In our rush to accept the many benefits of the silicon chip, we have abdicated our responsibilities. We have let the inmates run the asylum.

When the inmates run the asylum, it is hard for them to see clearly the nature of the problems that bedevil them. When you look in the mirror, it is all too easy to single out your best features and overlook the warts. When the creators of software-based products examine their handiwork, they overlook how bad it is. Instead they see its awesome power and flexibility. They see how rich the product is in features and functions. They ignore how excruciatingly difficult it is to use, how many mind-numbing hours it takes to learn, or how it diminishes and degrades the people who must use it in their everyday lives.

.....

The programmer wants the construction process to be smooth and easy. The user wants the interaction with the program to be smooth and easy. These two objectives almost never result in the same program. ...

The key to solving the problem is interaction design.

.....

All it requires is the judicious partnering of interaction design with programming. ..."

Propositions

I want to describe my position by a number of propositions which can be explained in more detail at the seminar and which hopefully will trigger interesting discussions.

Proposition 1: Co-evolution

Use contexts, computer-based artifacts, and their production contexts are co-evolving. Ideas about humans, tasks and the organisation of work (about the use context) influence the development of artifacts. Available artifacts influence humans, tasks and organisational forms. There is no stable state, the two worlds are learning all the time. It is an interesting question how learning can be supported best and how the exchange between the two worlds can be supported (Particpatory design? Users as designers?)

Proposition 2: Two faces, two interests

Interactive software has two faces. Software in use is in the background ("working through the Interface", S. Bødker). It is an enabler of work tasks. Users are not interested in the artifacts as such. They want to have a comfortable environment which fits their interest and tasks.

Software under construction is the complex material in focus for the software engineer. They struggle with complexity. Comfort for users usually means more complexity for software engineers. Software engineers are interested in keeping it simple, often at the cost of usability. Serving all user groups with the same software may lead to unnecessarily complex software for both sides.

Proposition 3: Artifacts express world views

Computer artifacts inevitably express the developers' images of the use context, especially their ideas about humans, tasks and work organization. These images are incorporated in the conceptual model of an application, the form of interaction and the interface available for the user (the Gestalt). The images are often too simplistic and mechanicstic. Users are restricted by these views, but can transcend them by innovative uses of the artifacts.

Proposition 4: Value of theoretical frameworks

Theoretical frameworks for describing and analyzing social aspects related to software development and use are helpful for understanding artifacts under construction and in use. They give little help for the design of artifacts. It would be interesting which frameworks help to understand better the software construction side, the attitudes of software engineers and their use of power.

Proposition 5: Power of software engineers

People good at technical construction (software engineers) are not necessarily good at design of artifacts. In most software projects they can decide on design issues - either because the artifact is not really designed at all, or because software engineers have the power to make final decisions on design issues influenced by their difficulties with managing the complexity of the construction.

The situation is similar to building houses without an architect. Software engineers are similar to civil engineers and construction workers. They have difficulties to imagine users who are different from themselves. They have special technical interests different from user interests.

Why do software engineers and programmers have and use this power?

Proposition 6: Need for interaction designers

We need use-oriented interaction design as a link between the context of use and the context of software construction. People working in this field need knowledge about use contexts, construction possiblities and design. They must be creative, inventive, sensible in the interest of users. They will design conceptual models of the application, forms of interaction and only after that the concrete interfaces. They will observe artifacts in use.

Proposition 7: Professional Interaction Design

Interaction design should become a profession. Interaction designers must have equal rights with construction and must precede it. Computer science departments should set up training programs for interaction designers together with other disciplines. Social frameworks can be taught to interaction designers to deepen their understanding of contexts of use and construction.

Proposition 8: Interaction design as a new challenge

The designers of interactive artifacts can learn from other design disciplines (product design, graphic design, media disciplines, ...), but they address a design problem of its own since the material used (software) is different (flexible stuff, complex systems, tendency to become too restrictive) and the design space is much bigger.

Artifacts must be stable and open at the same time. Stability is necessary to keep the software in the background when doing routine work. Openness is necessary to allow co-evolution. It must be possible to adapt/tailor the software by users and to change it by engineers if greater adaptations are necessary. This puts a new burden on designers and engineers: design and construct for stability and change at the same time.

It puts a new burden (and challenge) on users as well: use a more complex interactive system, learn to tailor, tailor it. Maybe groups of users have better chances when cooperating for adaptation.

The Use of Process Models in Software Development

Kari Rönkkö

Below follows a short discourse over the first thought that came to my mind in this writing moment when thinking about Software Development and the use of Process Models. At this conference that thought will represent my SOSO paper. As described below the thought is pointing at one aspect that a Process Model should deal with, especially in larger and/or distributed software development projects.

Process Model in larger and/or distributed software development projects.

Software Development as work differs from a lot of other design work insofar as the object to be designed is not visible. The object of design has to be represented in some way, this is done through the use of both persistent representations such as documents, diagrams, mock-ups…etc, as with help of more flexible and fleeting means of representations, such as utterances, sounds, and enactment when ‘talking design’ [1]. Together the persistent representations and the more flexible and fleeting kind of representations describes aspects of a later software. These are aspects grown out of collaborative design efforts that are collectively constructed by software developers. A bit clumsy it could be said that during software development the aspects of the later software are captured in two kinds of more or less lasting mediums. There are ‘visible signs’ left behind as the results from software developers collaborative design efforts, the documentation. Then there exists ‘hidden signs’, meaning the ‘collectively constructed agreements’ that give ‘life’ to the ‘visible signs’ such as the documentation. In software development projects with more than one developer involved the constructing of the ‘hidden signs’ or ‘the agreements’ is relying on the use of the more flexible and fleeting means of representation. Depending on different degrees of involvement in the actual work of creating the visible signs there also comes to exists different levels of inter subjective understanding of the later software that are under construction. Peter Naur addressed this issue already 1985 in his article ‘Programming as Theory Building’ where he argued that the development of a theory are relating the design of the interface and the software itself to its anticipated use [2].

Problem tend to arise in software development projects that are large and distributed, one reasonable explanation could be that both kinds of ‘signs’ or mediums representing the later software as described above are intertwined and depending on each other. An intertwining that tends to be difficult to achieve in large software development projects, as well as in distributed software development projects. Perhaps because of the fact that the use of the more flexible and fleeting means of representation that eases the collectively construction as well as reconstruction of an later software has it’s price for the involved developers regarding both time and presence. Where does this leave us? There exist a need to either focus on the constructing of other means for representing the flow of argumentation that is needed to reach common agreements built upon shared ‘enough’ understandings, otherwise the existing flow of the more flexible and fleeting means of representations has to be taken more seriously. In the later case the structuring of that intertwining seems to be an issue for a Process Model.

References

Dittrich, Yvonne and Rönkkö, Kari. 1999 Talking Design: Co-construction and the Use of Representations in Software Development, Iris 1999

Naur, P. (1985): ‘Programming as Theory Building.’ *Microprocessing and Microprogramming* 15(1985), 253-261.

Technologies for Self-Organisation

Dan Shapiro

When people in our kind of community speak about designing computer systems to support work, we don't mean producing a better word processor. What we mostly do mean is something to do with supporting work process.

Despite serious hesitations and dilemmas about the largely unknowable political and societal consequences, it is the possibility of supporting work process which for us comes closest to unlocking the power of the machine, this reality-constructing machine. We want the machine to support us in action, so what could be more obvious than we should do so by getting it to act? We want to press the button and have it act for us, albeit of course always under our control – what Marku Nurminen very evocatively calls "sleeping labour", and what Christiane Floyd (in this report) also refers to as "auto-operational form".

Because this operation is in the machine, it must be modelled, and the model formalised. But, at least since Lucy Suchman's work, we have known how impossible it is to successfully model human action as a process – it is always situated, contingent, unspecifiable, unknowable, unpredictable. This is what makes the modelling for the machine a "wicked problem", meaning an under-specified one, as Geraldine Fitzpatrick points out (in this report). And it is why even the most sympathetic attempts to realise process in the machine often end up as "workflow" systems with all their pejorative connotations. For members of this community – though sadly still not much beyond it – this 'wickedness' is by now old news. Yet, like moths to the candle-flame, we are drawn back again and again to attempting the impossible, to trying to model human action, which of course means human social action.

Often, the next step is to call for an ethnography of work practice to solve, or at least address, this problem of "requirements capture" for process. However, for the purposes of design, ethnography is just as vulnerable to the above objections as any other method of requirements elicitation. Ethnographic studies can, I believe, fairly claim to reveal fascinating and unsuspected aspects of work practice, and through this to change the climate and environment in which system design is attempted. But ethnography cannot magically change unspecifiable and unpredictable patterns of action into ones that can be known and modelled. This is because human action does not 'simply' realise an extremely complex plan (if that were so, ethnography or some other method might reveal it); rather, human action is produced anew, moment by moment, 'each new next first time' as the ethnomethodologists put it.

If we take seriously that we cannot expect to successfully design for human process, what possibilities are left?

We could limit ourselves to only addressing processes, perhaps such as some accountancy processes, which are already highly prescribed – in which, that is, practitioners have accepted that they will configure their practices as though constrained by a model.

We could refrain from attempting to do more than provide a set of 'basic' tools – a word-processor, an email package, etc. – leaving entirely to practitioners how to combine each use of these in process.

We could look to the customisation of tools – though experience is that while support for this is nearly always provided, it is rarely actually used.

We could look to do-it-yourself means of end-user programming such as macros, which Phil Kraft (in this report) refers to as tertiary programming – again a potentially very powerful support which seems in practice hardly ever to be taken up.

An approach which we developed in a previous project of 'bricolage': that is, instead of designing ab initio, to try to take advantage of the torrent of hardware, software and services now pouring out of the IT industry, and to combine, adjust and continually evolve these to fit with the evolving needs of the work practice. I believe this is a useful approach, and organisations must in fact do something like bricolage all the time, and of course not only of technology. But it is hard to envisage having an ethnographer and a participatory designer constantly on tap, so it would need to evolve some other methods. It is also more suited to a week by week, day by day, or even hour by hour cycle of evolution, than the moment-to-moment production of process which I have identified as the problem.

In trying to go beyond these possibilities I have been reflecting on some recent projects of ours and on some other work. I have been working with colleagues at Århus on a variety of projects for some time now, and my attention was caught by some features of the Århus Webvise hypermedia system. I was of course impressed by its capabilities specifically as a hypermedia system, such as its ability to link to endpoints within a document, its ability to follow links in both directions, its robustness when documents containing links are moved or changed, etc. But I was even more impressed by the philosophy of use which has evolved around it, that it should be possible to make links which would facilitate future work with extremely minimal overhead of additional work, because making the links would closely follow and reflect the normal process of doing the work.

Perhaps we can use this philosophy as a clue to another category of system, one which offers support to process not in a direct way but by providing for practitioners' self-creation of environments which are reflections of the work and which embody and retain the traces of its latest activities.

We hope that two systems which we are currently working on – Wunderkammer and Manufaktur – will turn out to have some properties of this kind. They are part of the ESPRIT DESARTE project on which we are working together with Århus and TU Wien, with user partners in architecture and landscape architecture.

The Wunderkammer is a system which is intended to support the inspiration, concept forming and concept maintaining aspects of aesthetic design in architecture and landscape architecture. It is based on observing the ways in which design professionals assemble around them and often decorate their working spaces with favourite artefacts, examples from previous projects, books, pictures, photographs, models, etc. etc. When working on a specific project, these are supplemented with more targeted collections of materials which have a more direct relation to the work in hand. There are great opportunities for this to be digitally augmented since, especially via the web, there is a potentially unlimited resource of such materials. But 'where' to put such a volume of primarily visual material in such a way that it can usefully be encountered, located and deployed?

The Wunderkammer has two linked versions, one in 2D and one in 3D, allowing many such possibilities to be explored. The 2D version uses a collage of a large number of panels which show or represent landscape types, assembled to form a larger landscape 'picture'. This collage was entirely produced by our landscape architect partners. It provides a visual "association space" for primarily visual materials, which can be 'dropped' into one or more panels. They pick up default keywords from the panels, which can be edited. Here, then, a very domain-specific environment is being created by practitioners themselves in the course of their work and with very little compulsory overhead. There are various ways for the materials in it to be systematically found or to be 'encountered' – the possibilities for the latter are especially rich in the 3D version. Of course, all the 'conventional' ways of finding digital material – by filename, by location in the file structure, by date, etc. – still remain.

The Manufaktur is intended to support a wide range of practice in aesthetic design. It is based on observing the crucial role played in the work of design by the manipulation, arrangement and juxtaposition in physical space of all sorts of materials – plans, pictures, texts, tables, samples etc. One's first thought is that this is preparatory to doing the work, but it becomes very evident from observation that instead this activity importantly constitutes the work.

The Manufaktur is a collaborative 3-D workspace which helps design professionals to create and maintain the context of a task or project, and to act within that context. It allows multimedia documents to be placed in the 3D space, and to be oriented in 3D at their locations. They can be grouped and linked. They can be made more or less transparent, and they can be 'illuminated' with different coloured light. The workspace can be given a topography, e.g. to provide a 'rooms' metaphor. It can be shared and saved. Documents can be worked on in their native applications, and their Manufaktur representation is updated in near-real-time through OLE/ActiveX links.

This is therefore also an environment which provides a context and support for work and which embodies a trace of its activities, without requiring any additional overhead to maintain it – working in the environment is constituting it as a support. There can be any number of Manufakturs, with overlapping or partially overlapping contents. This is therefore a hugely powerful resource for supporting a range of activities. Its properties will, however, call for much investigation, since it in effect allows practitioners, individually and collectively, to work in any number of 'parallel universes'. It also significantly extends important issues, which already exist, about the relationship between physical and digital manifestations of work materials.

I end with the open-ended questions of:

whether this perspective, of providing support for practitioners' self-creation of environments which are reflections of their work and which embody and retain the traces of its latest activities, is a useful direction to follow; and

if so, what other forms we can invent and design.

More details on the Wunderkammer and the Manufaktur are provided in:

M Büscher, M Kompast, R Lainer & I Wagner, "The Architect’s Wunderkammer: Aesthetic Pleasure & Engagement in Electronic Spaces", Digital Creativity 10/1, pp. 1-17, 1999.

M Büscher, P Mogensen, D Shapiro & I Wagner, 'The Manufaktur: Supporting work practice in (landscape) architecture', Proc. ECSCW'99, The 6th European Conference on Computer Supported Cooperative Work, Copenhagen, 12-16 September, 1999.

Acknowledgements:

I am very grateful to my colleagues on the DESARTE project for many kinds of contribution: Monika Büscher and Liz Hodgson at Lancaster; Liz Ashburn, Walter Truscott and others at SGS Environment; Preben Mogensen, Michael Christensen and Klaus Hansen at Århus; Ina Wagner, Martin Kompast, Florian Lederman and others at TU Wien; Rüdiger Lainer and others at Architektbüro Lainer in Vienna; and members of our industrial partners.

Position paper

Jesper Simonsen

My background is rooted in the Scandinavian tradition for Participatory Design (PD). I am employed as an Associate Professor in Computer Science at Roskilde University. Prior to joining the University, I worked a number of years as an IT designer within Danish industry. My main research interest is the study of work practices of users and designers for the purpose of offering theories and methods for systems design in an organizational context.

Since 1991, I have worked within the MUST-research program along with two of my colleagues, Keld Bødker and Finn Kensing. Within this research program, we are currently finishing a textbook describing the MUST-method. The MUST-method is concerned with the early stages of systems design: The idea behind the method is that users and their managers in a specific organization can decide which IT systems are needed and relevant, and how they can be provided, i.e. purchased, and/or developed, and implemented. During the coming 4 years, I will be involved, together with 15 Danish researchers, in a new research program, DIWA – Design and use of Interactive Web Applications.

In an organizational approach to PD, technical aspects are seen in their organizational context. This means that organizational aspects like work processes, work organization, like skills, qualifications, and competencies, like organizational cultures, policies, strategies, and politics, and like latent and manifest conflicts, etc., become relevant and important to the design process.

The MUST-method aims at taking organizational aspects seriously by combining design intervention with the use of ethnographically inspired techniques, such as unobtrusive and participant observation, (in-situ) interviews, thinking aloud experiments, and audio or video recordings. The method introduces these techniques to the designers, in contrast to other approaches where ”real” ethnographers cooperate with designers.

As pointed out in the MUST-method above, I take a highly pragmatic approach to software development, work, and organizational change. I belong to the generation of researchers following the Scandinavian PD union/researcher projects with the political ideology of enhancing democracy at work. In my opinion, the criteria for success in normative research within design must be that the research results are industrially applicable also in a relatively short term horizon. This involves the following:

- That researchers develop their own experiences with new design approaches by action research and by playing the role of designers themselves.

- That the researchers take their starting point in today's practice of designers, try to understand the conditions of an industrial and commercial design practice, and develop, align, and apply new approaches to this practice.
- That this research influences practice – in the short term horizon – by proving its value (for example through experiments in cooperation with designers) as well as – in the long term horizon – through educating new generations of designers.

References

MUST-research program, URL: http://www.must.ruc.dk

DIWA-research program, URL: http://www.diwa.dk (active from September 1999)

My home-page, URL: http://www.ruc.dk/~simonsen

Simonsen, Jesper: *Designing Systems in an Organizational Context: An Explorative Study of Theoretical, Methodological, and Organizational Issues from Action Research in Three Design Projects*, Ph.D.-thesis, Writings on Computer Science No. 52, Computer Science Department, Roskilde University, Roskilde, Denmark, 1994.

Simonsen, Jesper, and Finn Kensing: "Using Ethnography in Contextual Design", *Communications of the ACM*, Vol. 40, No. 7, July,1997, pp. 82-88.

Simonsen, Jesper, and Finn Kensing: "Make Room for Ethnography in Design! *ACM-SIGDOC, The Journal of Computer Documentation*, Vol. 22, No. 1, February, 1998, pp. 20-30.

On requirements using ethnography and the scaling up of these techniques

Chris Westrup

The use of ethnography and, more recently, video based techniques as aids in requirements analysis is widely discussed. Both these techniques indicate a welcome move towards an awareness of the complexity of social life in organisations and the difficulty in developing computer systems in many circumstances. However the purpose of this paper is to draw attention to some of the difficulties in such approaches and to stimulate discussion on the limitations as well as the benefits of such techniques.

Ethnography and video analysis can be seen as projects based on realism: there is something that can be collected about the organisation and these techniques are sophisticated devices to assist in this task. In summary, there are three issues that I think need to be addressed:

Ethics

One argument is that the information collected using these techniques is the property of the informants. This is becoming accepted in social anthropology where codes of practice have been drawn up to regulate the use of ethnographic information by ethnographers so that informants retain rights over the collected material. In a work setting, what ethnography and video analysis attempt to reveal are the working practices which enable work to be performed. Using Marx we could argue that ethnography could be, yet one more, sophisticated approach to identify and expropriate the added value of labour by identifying the skilled practices of workers and incorporating much of them in technology. Even, if we consider, that the aim of ethnography is the support of skilled work, this issue still remains if those who work is being represented do not have complete control over the use of that material.

The role of the ethnographer/video analyst

Bauman made an interesting distinction between legitimation and interpretation as roles for intellectuals. Legitimation is maintained through the application of the expert knowledge in analysing situations to improve control. Interpretation recognising that competing interpretations of any situation are possible and seeks to identify differing perspectives and develop dialogue. It seems to me that ethnography and video analysis can be used either for legitimation or for interpretation: legitimation when work practices in one context are seen as exemplars for the use of technology elsewhere. For example, the use of ethnographic information on a new computer package in one work setting to then adjust the package to for other sites. Interpretation may occur when users are actively involved in discussing and reviewing the information collected by ethnography and video analysis. Ignoring such a distinction is tantamount to ignoring the power/knowledge conditions in any particular work setting.

Representation

Arguably, the purpose of ethnography and video analysis is to produce a representation of the setting which can be taken elsewhere and used to assist in the design or redesign of technologies. Though many may be concerned with the vast amounts of information produced, the question needs to be asked as to the limits of these representations. Under what conditions was it produced; with whose approval; for what explicit purpose; how much of the work setting is represented; is homogeneity present and why? And so on.

Each of these issues has important practical consequences. First, they question the ability of software developers to appropriate social science techniques. And second, it opens up for scrutiny how the insights derived from ethnographic inquiry in one organisation/social setting can be scaled up through the development of a computer package so that it may be applied in a multitude of organisations.