By The Way, Did Anybody Study Any Real People?

Susan Gasson, College of Computing & Informatics, Drexel University
Working Paper

Please cite this paper as:
Gasson, S. (2004) 'By The Way, Did Anybody Study Any Real Users? ', Working Paper. Last updated 03/28/2006; retrieved <insert date of retrieval here> . Available from http://cci.drexel.edu/faculty/sgasson/papers/real.pdf

ABSTRACT

Despite continuing debates about the "user" emphasis in HCI, new design approaches, such as interaction design, continue to focus on humans as ***technology users***, constraining the human-centeredness of design outcomes. The paper reviews popular approaches to user-centered design and concludes that these methods are targeted at the closure of technology-centered problems, rather than determining suitable changes to a system of human-activity that require computer-based support. A dual-cycle model of human-centered design is presented that combines systemic inquiry methods with user-centered implementation methods. The paper concludes with a summary of the limitations of user-centered methods and a discussion of when these methods are appropriate.

Keywords: Design Methods, Human-Centered Design, User-Centered Design, Interaction design, Participatory design, Socio-technical design.

Introduction

By focusing on usability, the field of HCI too often overlooks the social context of ***use***. Bjorn-Andersen [3] criticized the narrow definition of human-computer interaction in the HCI literature, with the words: "it is essential that we see our field of investigation in a broader context. A 'human' is more than eye and finger movements". So how ***do*** we design for human-centeredness? Gill [14] defines human-centeredness as "a new technological tradition which places human need, skill, creativity and potentiality at the center of the activities of technological systems." The human-centered approach to the design of technology arose as a reaction to perceptions that traditional approaches to technology design deskill technology users and impoverish the quality of working life. While many of the issues of human-centeredness have been taken on board by the HCI field, many have been considered to lie outside the boundaries of "user" interactions with computers. This paper argues that the continued focus of HCI on the concept of a technology user is counterproductive and constrains the human-centeredness of system outcomes. Despite continuing debates about the user

focus, this issue has not gone away and continues to constrain new HCI approaches, such as interaction design [10, 28, 32].

There is a wide body of literature on the development and application of human-centered systems in other fields, such as the organizational literature on the impact of technological change at work. Some of the main ideas arising from the human-centered design literature are:

- The human-centered approach rejects the idea of the "one best way" of doing things: that there is one culture or one way in which science and technology may be most effectively applied [14]. Human-centered design advocates the design of flexible systems that permit the people who work with them to shape and manage their work [20].
- Technology is shaped by, and shapes in turn, social expectations: the ***form*** of technology is derived from the effect of these social expectations upon the design process [22]. Human-centered design advocates the design of systems that question normative expectations of technology [21].
- The human-centered approach is opposed to the traditional, technology-oriented approach, which prioritizes computer-based information processing and technology-mediated communications over humans and their communicative collaboration [14].
- Human-centered systems production should concern itself with the joint questions of "What can be produced?" and "What should be produced?" The first is about what is technically feasible, the second about what is socially desirable [14].
- Objective and subjective knowledge cannot exist independently of each other. The explicit, rule-based knowledge needed for computer-based systems is useless without the "corona" of tacit and skill-based knowledge which surrounds the explicit core and through which explicit knowledge is filtered [9]. Human-centered design acknowledges the need for ***informal*** information systems that enable the use and communication of implicit knowledge.
- A human-centered approach avoids the prevailing tendency to separate "planning" tasks from "doing" tasks, when designing new systems. This separation results in deskilled human technology users who are less equipped for exception-handling and whose quality of working life is impoverished and uninformed [9]. Human-centered design strives for socio-technical systems that support meaningful and enriched work [14, 21].

Human-centered design takes a socio-technical view of an information system, focusing on two "worlds" of inquiry:

The social system of interacting human activities, multiple, implicit (and often conflicting) goals, human understanding and knowledge, business context and application-specific cultures and practice.

The technical system of formal, rule-based procedures and technology, managed by performance indicators and exception-handling.

Any effective human-centered design approach should focus on balancing the requirements of these two worlds. A theme ignored by much of the HCI literature is that the ***forms*** of available technology have an influence on the range of social choices available in work design [21, 29]. The form of a technical system determines the effect of that technology upon its human users,

regardless of how well the interfaces to that system have been designed. This idea has been taken up to some extent by recent developments in HCI - in particular in Interaction Design [32]. But the way in which systems are conceptualized in the HCI literature is still limited by its traditional interests.

The Nature Of Information System Design

Design, from an information system (IS) perspective is alternately classified as a single stage in the technical systems development life-cycle [19], as a fundamental part of technology conceptualization and implementation throughout the whole system development life-cycle [12], as reuse from a repertoire of technology exemplars [22], or as “the interface between understanding and creation” [34]. Such diverse definitions of design are linked by two elements, according to Winograd:

" Design is also an ambiguous word. Among its many meanings, there runs a common thread, linking the intent and activities of a designer to the results that are produced when a designed object is experienced in practice." [33, ***page v***]***.***

Design is not an activity that can be separated from system investigation. We are designing a new system in our heads from the very moment that we perceive that a new piece of software, or a new information system, is required. But it is the joint focus on Winograd's two elements - process and product - that makes design so problematic. Goal-directed methods prescribe that the requirements for a successful outcome should be fully determined before implementation starts and many user-centered design methods attempt this approach. But if we examine the nature of the design process, it is strewn with discarded attempts to predict the experienced outcome. Some of these attempts will be successful and some will be forgotten by the time that the design is complete. Design is also an integral part of system implementation. To separate these stages is to offer up our hopes for the system on the altar of technical expediency.

Empirical studies of design show a progression in how we understand the design process. Studies in the 1970s and early 1980s view design from the initial perspective of hierarchical decomposition proposed by Alexander [1]. Such studies are prescriptive in their observations and tell us little about the nature of design [31]. Later studies inform us that technical designers do not work from problem to solution, but rather define both interactively [12, 15, 30, 31]. Turner [30] argues that “requirements and solutions migrate together towards convergence”. If the designer cannot supply a partial solution to an aspect of the problem, they reframe aspects of the problem to fit known solutions [15, 30]. Moreover, there is very little user-involvement in most system design projects, even when those projects claim to use "participatory" design [12, 18].

Design is not a structured, decompositional process, but a process of jointly conceptualizing system form and outcome. Designers frame the problem by drawing on repertoires of

previously-encountered, partial solutions. Problem and solution are conceived together and are inextricably intertwined. Problem definition is guided by the designer's experience of, or exposure to, suitable ***forms*** for a solution [15, 30]. If we start by focusing on technology use, we will get a technical solution. If we start by focusing on how people work, we will get a socio-technical solution.

System adaptability and use are constrained by the extent to which designers view the design process as planned and programmable, or emergent and evolutionary. Information system use is situated, not just in the work-context and tasks of a single user, but in the social worlds and multiple communities of practice to which that user belongs [5]. The overall process of design cannot be goal-directed and planned, because detailed goals and design strategies emerge through the process of design [5, 15]. Thus, IS design is not amenable to detailed planning. Approaches to IS design must necessarily allow for the emergence of new directions, driven by contingent understandings of system form and purpose. IS development processes are situated in the sociocultural context of the group and knowledge about how to develop an information system is not necessarily transferable to different situations [5].

Given this understanding, we now examine the contribution of HCI to design approaches in use and suggest some ways in which these might be supplemented by a more systemic perspective of the multiple user-worlds that require understanding for the effective design of computer-based information system technology.

Developments In User-Centered Design

Prototyping and Participatory Design

There are many reasons why it is difficult to demonstrate the contribution of user participation to a successful design outcome. Kappelman & McLean [18] contend that it is not so much user participation, as user ***involvement*** which leads to a successful outcome: that "state of psychological identification with some object, such that the object is both important and personally relevant". Kirsch and Beath [19], in an interview-based study of eight development projects at seven large, US firms, distinguish between:

Token participation, where users appear to be active participants, but technical designers supply domain knowledge and control feature selection.

Shared participation, where domain knowledge comes exclusively from users, who control the selection of system features.

Compliant participation, where technical designers educate users in the technical domain and convince users of the need for a technical solution which addresses the wider requirements of the firm, rather than local, user-centered needs.

Kirsch and Beath [20] were unclear about whether the outcome of compliant participation actually benefited the user, but this form of participation appears to be the dominant approach in corporate systems development. It is also a ***prima facie*** example of the exercise of conceptual power [23]. Conceptual power is exercised by technical designers when they manage user perceptions of how a technology can be employed, or when they develop IT policies and procedures that constrain user choice.

In the middle, we have shared participation, or participatory design (PD). Muller et al. [24] list a variety of methods for PD, classified by the ***position of the activity in the development cycle*** and ***who participates with whom in what.*** The latter axis ranges from "designers participate in users' worlds" to "users directly participate in design activities". For PD to be participatory, user-worlds must be effectively represented in the design. But user-worlds are often inadequately represented because of cost constraints, or a lack of appreciation of the significance of users' perspectives. True participatory design approaches, involving users in the evaluation and definition of system possibilities are extremely rare, as the additional expense and timescale are often difficult to justify to corporate accountants and managers. We have ineffective methods to promote and to represent human collaboration in system models. As Gasson [13] discovered, user worldviews are all too often relegated to "interface" considerations by technical system designers. The use of participatory design becomes a power struggle between, on the one hand, "rational", technical system designers and, on the other hand, "irrational" user-representatives who are unable to articulate system requirements in technical terms. The concept of empowering workers raises hackles: this is seen as "social engineering" that compares unfavorably (in scientific, rationalist discourse) with "software engineering". Designers who engage in such irrational behavior must have a subversive agenda that is counterproductive [25]. Thus, participatory design in all its forms is politicized and subsumed to the less intrusive (and much less confrontational) path of producing user-centered design "methods" that can be partially used, in ways that technical designers choose.

Interaction Design

A more recent development of user-centeredness in design, from the HCI field, is that of interaction design. Interaction design considers a much deeper set of concepts than the traditional HCI interest of user-interface. But the meaning of "interaction" seems to vary with the author advocating it. Interaction design, as defined by Cooper [10] is "goal-directed design":

" Interaction designers focus directly on the way users see and interact with software-based products."
[10, ***page 16***].

Interaction design is therefore product and development driven: this approach defines what software system products should be built and how they should behave in a particular context [10]. But goal-directed approaches are only appropriate when the problem is relatively well-defined [7]. Most organizationally-situated design goals are emergent. Yet a similar, goal-driven

approach is taken by Preece et al. [28], who emphasize "the interactive aspects of a product" (***page 11***). Although they extend the goal-driven concept with rich discussions of use, their perspective is also essentially driven by the notion that design is centered around conceptualization of a computer-based product with an individual user. Inquiry into the socio-cultural worlds of its use are secondary. Winograd [32] defines interaction design very differently:

" My own perspective is that we need to develop a language of software interaction - a way of framing problems and making distinctions that can orient the designer. … There is an emerging body of concepts and distinctions that can be used to transcend the specifics of any interface and reveal the space of possibilities in which it represents one point." [32, ***pages 8-9***].

So interaction design has the potential to consider a space of possibilities, but in general is constrained to specific interactions with a predetermined technology by the tradition of HCI discourse. This discourse ***starts*** with a concept of "the computer" (or computer-based technology) and only then considers the context of the human-computer interaction. This has the effect of moving the design model back to the historically ***unitary*** focus of HCI: a single technology user, moving towards closure of a single, task-related problem, in isolation from the social world of work that surrounds them. Interaction is thus reduced to interface.

HCI research into user-centered design has, however, had a significant impact on software development ***practice***, as evidenced by the emergence of two schools of design methodology: the production of Use-Cases as part of Uniform Modeling Language (UML) approaches to system design and "agile" software development.

Use-Cases in UML

The production of use-cases [17] has now been absorbed into the Unified Modeling Language [4] approach to formal system representation and modeling. The primary concerns here are the correctness and completeness of a technical system model. Use-cases constitute a representation of ***interactions*** between different classes of user and a computer system. From these use-cases, formal object-oriented models and specifications may be defined, that enable the production of a technical system. A example of a use-case diagram is shown in Figure 1. Special cases (extensions) of associations between objects or business processes are shown with a dotted line, while normal associations are shown with a solid line. So in the example given here, the credit limit would only be checked in some circumstances (e.g. if the account balance is insufficient for the withdrawal).

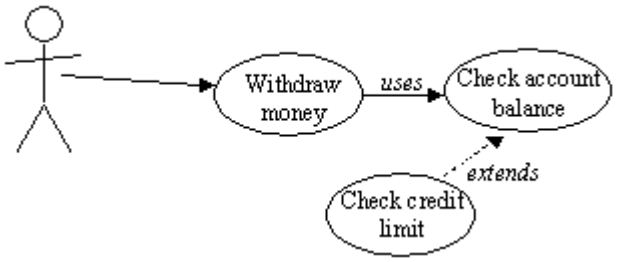

Figure 1: Example of a Use-Case Model

A benefit of the method is that designers are encouraged to base use-case models on the viewpoints of multiple stakeholders. A high-level model is thus constructed that , ostensibly, starts with the user requirements of the proposed system and develops a set of "business" processes that the system will (partly or wholly) automate.

In HCI terms, this method represents a major victory for user-centeredness in technical practice. System conceptualization starts with an understanding and definitions of user-interactions. But is this really true? The use-case model focuses on the articulated requirements of a single user. It has no way of surfacing (or even recognizing the existence of) implicit requirements. Where multiple viewpoints are sought, the task is to reconcile these, not to ***represent*** (often conflicting) user requirements of the system. Additionally, most use-cases appear to be produced by the designer imagining how users would interact with their target system. Use-cases are largely based on short interviews and there is little opportunity for validation. So we get back to the problem identified by Norman [27]:

" The designer expects the user's model to be identical to the design model. But the designer does not talk directly with the user - all communication takes place through the system image." [27, ***page 16***].

But the most serious problem with this approach (from a human-centered perspective) is that the design of user-interaction starts with a concept of the computer system. Designers gradually build up a set of interactions with a technical system whose form is preconceived (as demonstrated by their ability to imagine and represent interactions with this system), rather than with a conceptualization of users, their work, their communities of practice, the problems that they face, and their lifeworlds. An understanding and definition of system interactions are therefore formed by the designer constructing a model of the system as a set of functions, rather than by an understanding of the needs of potential users and other system stakeholders. So, while system design based on use-cases may be considered user-centered, it does not fulfill the requirements for human-centeredness.

Agile Software Development

A recent practitioner initiative in user-centered design is "agile" software development. This approach recognizes that the separation of software design and construction is artificial and

counterproductive (see the section on the nature of design, above). Agile software development treats the production of computer-based systems as a series of short, phased deliveries, combining iterative prototyping with user-participation and evaluation.

Highsmith's [16] ***Adaptive Software Development*** is a practitioner approach that operationalizes many of the user-centered design ideas from HCI research. This approach shares the minimalist agenda of eXtreme Programming as a way of coping with ever decreasing product life-cycles and rapid technical change. Highsmith rejects what he terms "monumental software development", in favor of "fitting the process to the ecosystem". Highsmith argues that systems design should respond to the contingencies of the local context, rather than fitting the problem analysis to the framework underlying a formal analysis method. ***The Agile Manifesto*** [11] values:

Individuals and interactions over processes and tools.

Working software over comprehensive documentation.

Customer collaboration over contract negotiation.

Responding to change over following a plan.

eXtreme Programming [2] is an increasingly popular form of agile software development. Based partly on the concept of scenario analysis [6] - a concept that is familiar to HCI researchers but novel to many technical system designers - the ***eXtreme Programming*** design approach subsumes formality of method to expediency. Technical systems developers work in pairs with selected users, to generate short scenarios, which are coded into a system prototype. One developer codes, while the other checks the code for authenticity and correctness (these roles are swapped frequently). The user is invited back to validate the prototype against the scenario and to generate additional scenarios, based on their realization of shortcomings or omissions in the original scenario generated, after having used the prototype.

This is not to attack the approach: it is a laudable attempt to deal with problems of implicit knowledge, evolutionary learning (by users) of what technology has to offer for their work, and misunderstandings between technical designers and users, as technologists gradually enter the lifeworld of the user. But the approach is based on a very small selection of "users", there is no attempt to understand or investigate the wider, socio-technical system of work and there is little attention paid to the selection of appropriate "users" to generate scenarios. Additionally, it suffers from a common problem of evolutionary prototyping: the approach starts with the specific intention of building a technical system, not with the intention of bringing about organizational and technical change.

In its focus on emergence and "the people factor", agile software development may be considered human-centered in its intent. However, its lack of techniques and its ultimate emphasis on the practice and profession of producing software systems renders it vulnerable to

deadline-driven expediency, as demonstrated in the lack of common ground in a fascinating conversation between Kent Beck (the creator of eXtreme Programming) and Alan Cooper (an advocate of Interaction Design) [26], even though both of the participants took a product development centered view.

Soft Systems Methodology

Checkland's [7, 8] concept of "soft" systems analysis exposes human-activity systems to view in an holistic inquiry. Soft Systems Methodology (SSM) has been popular in the UK and Europe for many years. SSM is based upon a seven-stage method that permits the problem investigator and stakeholders to take a systemic approach to inquiry about appropriate change.

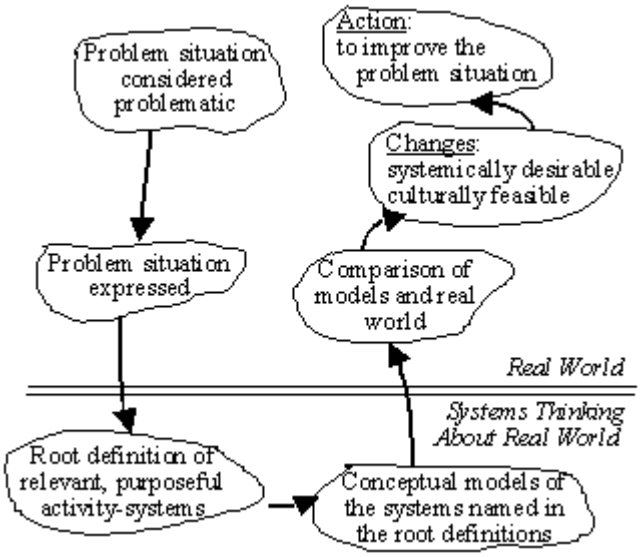

Figure 3: The conventional seven-stage model of SSM

While there is not space here to describe the soft systems approach to problem inquiry in detail, an overview may explain why this approach is so different from most user-centered approaches to system design.

Firstly, Checkland [7, 8] emphasizes the difference between "real-world" modeling and the production of models that explore possibilities for organizational change through systems thinking about the real world. SSM provides a way of progressing from models of how the organizational "world" works now, to models of how it could work, to derive desirable and feasible actions for change. SSM does not deal with technology as a design object, but only as a single component in multiple actions for change [8]. Therefore technology design follows systemic information system design, rather than guiding and constraining it.

Secondly, SSM provides techniques for the surfacing of implicit aspects of work and social practice. By requiring stakeholders to jointly define sequences of activities that could constitute

an improved way of performing work, SSM exposes to view many implicitly-held models of how the world works now and how it could work.

Thirdly, SSM provides explicit recognition of the multiple and often conflicting needs of various stakeholders [7, 8]. By recognizing that work problems are interrelated, Checkland provides us with methods to disentangle the mess. In SSM, different "systems" of change are modeled separately, with problem-definitions resulting from input from ***all*** the stakeholders (not just users) of the new system. Checkland [7] emphasizes a focus on the ***various "***customers" of the system, whom he defines as the "victims or beneficiaries of the proposed actions for change". Problem-definitions are then presented for negotiation among system stakeholders.

Fourthly, the problem investigator, or "would-be improver of the problem situation" is not cast in the role of ***expert***, dispensing solutions from a vast fount of technical experience, but as ***facilitator*** , providing support to organizational actors who themselves define a system's purposes and functions. The role of facilitator necessitates debate around differing perspectives on the system's role, which is not the same as seeking consensus (the implicit aim of technology-centered systems analysis and design). Consensus too often means that important objectives of the system are sacrificed to organizational politics, or that system design requirements are "rationalized" by a technical system designer. SSM then, provides a means of making conflicts of interest open to debate and explicit, reflecting social and communicative practice.

Whilst SSM is an excellent approach to systemic problem inquiry, it does not explicitly deal with technical system design, but with a set of actions for desirable and feasible change that define the purposes and roles of computer-based technology. It provides an excellent way of opening up the systems problem for inquiry; it does not provide ways of closing it down. For these, a more traditional user-centered technology design approach would be appropriate.

Design As Mutually-Interacting Inquiry and Implementation

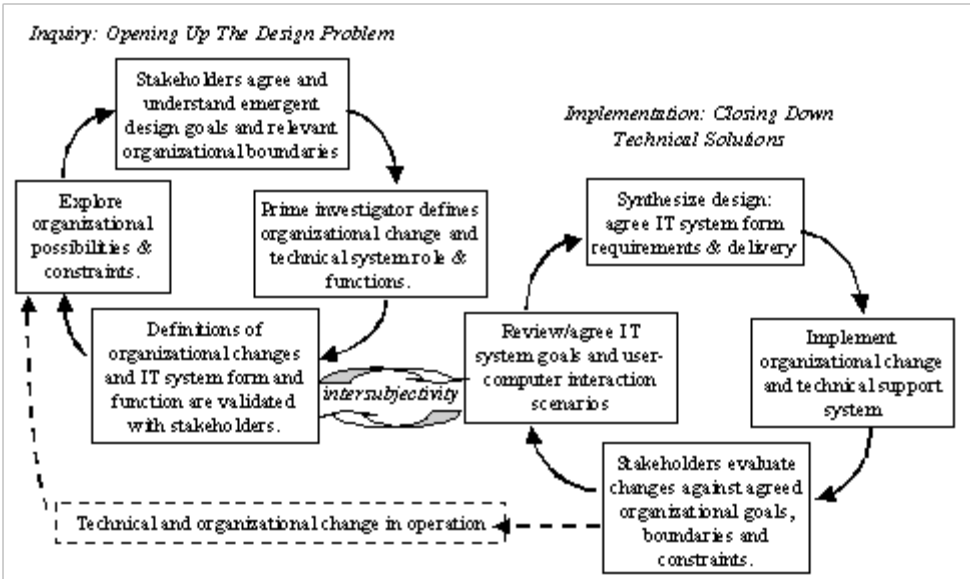

Figure 2: Dual-Cycle Model Of Human-Centered Design

It can be seen from the discussion here that most user-centered approaches are concerned with ***closing down*** a technology-centered and goal-directed IS problem- definition, not about ***exposing*** (or opening up) the problem situation for examination and debate. Even in agile software development, the design "problem" is determined very early in the process and remains unexamined after that point. The generation of user-scenarios may change the way in which the technical system is implemented, but it does not change the essential form and role of the technical system. Even participatory design approaches do not provide ways of questioning the initial problem-statement, but focus on usability and task-support. User-centered approaches to the design of technology do not change the fundamental nature of the "circular" system development life-cycle. They merely "rotate" the life-cycle through 90 °, so that the cycle is driven by user-evaluation of system design requirements, rather than by technical evaluation of system design requirements. This rotation does not question many of the essential contradictions of the traditional perspective, because it inherits the "problem closure" life-cycle emphasis. While this focus on problem closure may have been appropriate at a time when IS designers were concerned with relatively well-defined, technical problems, it is no longer appropriate for the human-centered design of complex, organizational information systems.

To resolve these problems, design may be managed as shown in Figure 2. (This model is derived from empirical studies of IS design in context [12, 13]). It represents two "cycles" of the design process, to deal separately but interactively with system inquiry and implementation (opening-up of problems and closing-down of solutions). The dual-cycle model emphasizes the importance of systemic inquiry as part of the design process. The problem closure models

within which we normally work delegitimize user participation and prevent users from revisiting problem definitions [12, 13]. This model acknowledges the process of involving stakeholders in defining actions for organizational and technical change, as part of a procedural design approach. Although the implementation stage is treated almost superficially by this model, this is not to claim that the implementation of technical or organizational change is unproblematic. But we already have good methods to manage this stage, resulting from HCI research and practice, as discussed above. However, problem inquiry and definition are insufficiently researched, in the way in which we approach design.

To overcome the "milestone" problem (i.e. convincing accountants to fund a project with no deadline in sight), a set number of relatively short inquiry-implementation cycles should be planned for. A suitable method for the inquiry cycle of this model is Soft Systems Methodology [7], discussed below. This method has the advantage that it fits well with the business process perspective with which managers are familiar (and so less open to challenge from the "rational" perspective discussed earlier than participatory design), while encouraging stakeholder participation and exposing internal contradictions.

Selecting A Suitable Design Approach

Sometimes, a goal-oriented approach is appropriate. If you have a well-defined problem, which you can solve (at least in part) with known technology, goals are relatively easy to set. Whether or not you believe that goal-oriented design is appropriate depends on your perspective, which is normally informed by the types of problems that you deal with in your day-to-day work. The decision about which design method(s) to used depends on the familiarity of the technology and on the extent to which the IS "problem" can be defined.

The matrix shown in Figure 4 may be used to determine whether goal-oriented methods (and, by implication, most user-centered design approaches) are appropriate. The left axis of this matrix represents the extent to which designers of organizational and technical solutions have experience of similar problems. The top axis represents the extent to which the organizational "problem" (in all of its complexity) can be clearly & simply defined, by everyone involved in the problem situation.

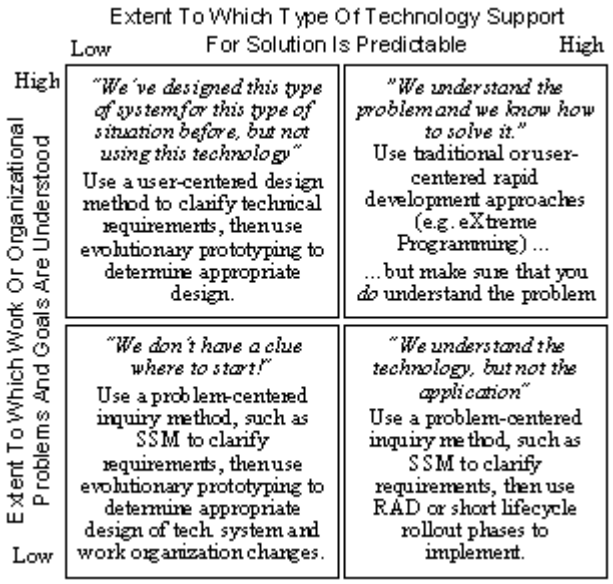

Figure 4: Selection Matrix For Design Approach

If stakeholders feel that the organizational "problem" is well-defined and agreed, then user-centered methods are appropriate for the design of that part of the solution which is computer-based. How they are used depends on how familiar designers are with the technology to be used (see Figure 4). If stakeholders cannot agree a suitable problem definition, then complex inquiry methods are required to provide such a definition. These methods should not be treated as a "one-off" process, but should be used cyclically, as shown in the dual-cycle model of Figure 2.

Summary and Conclusions

While HCI-inspired design approaches constitute a major advance over traditional approaches, they share five limitations with traditional design approaches:

Technology-centered design. Even "user-centered" design is focused on an understanding of interactions with a technical system, not on the socio-cultural, evolving and interrelated systems of human activity that comprise organizational work.

Goal-directed process planning. We have discussed how goals evolve with inquiry and through the process of design. Any effective design approach must recognize and enable its processes to be guided by emergent and incomplete design goals, unless the design problem is exceptionally well-defined (see Figure 3, below).

Privileging of explicit "requirements" of system change. Most user-centered design approaches have no way of surfacing the implicit models of work held by system users and

other stakeholders. System designs therefore represent only those rule-based aspects of organizational work that people can articulate.

Focus on the stakeholder as system "user". This has the following consequences:

- This focus ignores the interrelated work-tasks and relationships of human beings, limiting system design models to a representation of a single individual interacting with technology.
- Human concerns may be relegated to considerations of interface rather than examining the role that technology does or should play in their work. This allows technical designers to marginalize user concerns, fitting the interface round the system, even when the design approach is "human-centered" [13].
- Technical designers are permitted to exercise ***conceptual power*** over users [23]. Technical "experts" direct user concepts of new technology and constrain their expectations of what the technology can or should do.

Focus on problem closure. Particularly when attempting to resolve the complex coordination and communication problems arising from social, collaborative work environments, we need complex methods of inquiry. These are not provided by "user" centered methods, whose focus is normally on a single user performing a single task.

In the spirit of inquiry, we conclude with a problem-statement and a proposed solution.

It is proposed that the two "worlds" of socio-cultural work and technology-interaction are incommensurable. We cannot analyze them using common methods, nor can we derive procedures and methods for producing software that satisfies the needs of both. HCI methods are targeted at closing-down technology-***centered*** problems, rather than opening up a technology-***supported*** system of human-activity for examination and change. In "user-centered" system design approaches, the boundary of inquiry is too limited for designers to consider aspects of context and socio-cultural significance that would make the system "human-centered".

What is required is a combination of systemic inquiry methods with user-centered interaction methods for the design of supporting technology. We should focus on using approaches and methods that permit us to operate in different modes of inquiry in each world and then use that unique, human quality that we all possess - the ability to synthesize across incompatible domains of knowledge - to produce satisficing solutions for the real world.

REFERENCES

- Alexander, C. (1964) ***Notes On The Synthesis Of Form***, McGraw Hill
- Beck, K. (1999) ***eXtreme Programming Explained: Embrace Change***, Addison-Wesley
- Bjorn-Andersen, N. (1989) 'Are 'Human Factors' Human?' in H.K. Klein & K. Kumar, Eds., ***Systems Development For Human Progress,*** North Holland
- Booch, G, Rumbaugh, J and Jacobson, I (1996) ***Unified Modeling Language User Guide***, Addison-Wesley

- Brown, J.S. & Duguid, P. (1992) 'Enacting Design for the Workplace' in P.S. Adler & T.A. Winograd, Eds., ***Usability: Turning Technologies Into Tools***, ACM Press, 164-197
- Carroll, J.M. and Rosson, M.B. (1992) 'Getting Around The Task-Artifact Cycle, How To Make Claims and Design By Scenario', ***ACM Transactions on Information Systems 10***, 2, 181-212
- Checkland, P. (1981) ***Systems Thinking, Systems Practice,*** John Wiley & Sons.
- Checkland, P. (1998) ***Information, Systems and Information Systems: Making Sense of the Field,*** Wiley
- Cooley, M. (1987) '***Architect Or Bee?: The Human Price Of Technology***', Hogarth Press, UK
- Cooper, A. (1999) The Inmates Are Running the Asylum: Why High Tech Products Drive Us Crazy and How To Restore The Sanity, Sams Publishing
- Fowler, M. and Highsmith, J. (2001) 'The Agile Manifesto', ***Software Development Magazine*** - online: http://www.sdmagazine.com/documents/s=844/sdm0108a/0108a.htm
- Gasson, S. (1998) 'Framing Design: A Social process View of Information System Development', in ***Proceedings of ICIS '98***, Helsinki, Finland, 224 - 234.
- Gasson, S. (1999) 'The Reality of User-Centered Design', ***Journal of End User Computing 11***, 4, 3-13
- Gill, K.S. (1991) 'Summary Of Human-Centred Systems Research In Europe', Part 1: ***Systemist (Journal of UK Systems Society) 13***, 1, 7-26; Part 2: ***Systemist13***, 2, 49-75.
- Guindon, R. (1990) 'Designing the design process: Exploiting opportunistic thoughts', ***Human-Computer Interaction5,*** 295-324
- Highsmith, J. (2000) Adaptive Software Development **:** A Collaborative Approach to Managing Complex Systems, Dorset House Publishing
- Jacobson, I. (1991) ***Object-Oriented Software Engineering,*** ACM Press
- Kappelman, L.A. and McLean, E.R. (1992) 'Promoting Information System Success: The Respective Roles of User Participation and User Involvement', ***Journal of Information Technology Management3***, 1, 1-12
- Kapor, M. (1996) 'A Software Design Manifesto', in T.A. Winograd, Ed., ***Bringing Design To Software***, ACM Press, 1-9.
- Kirsch, L.J. & Beath, C. M. (1996) 'The enactments and consequences of token, shared and compliant participation in information systems development', ***Accounting, Mngt. & Info. Technology6***, 4, 221-244
- Kuhn, S. (1996) 'Design For People At Work', in T.A. Winograd, Ed., ***Bringing Design To Software***, ACM Press, 263-279
- Mackenzie, D.A. & Wajcman, J. (1985) 'Introduction' to Mackenzie, D.A. & Wajcman, J. (eds) ***The Social Shaping Of Technology***, Open University Press, UK
- Markus, M.L. & Bjorn-Andersen, N. (1987) 'Power over users: its exercise by system professionals', ***Communications of the ACM29***, 6, 498-504
- Muller, M.J., Wildman, D.M. and White, E.A. (1993) 'Taxonomy of PD Practices: A Brief Practitioner's Guide', ***in*** M.J. Muller and S. Kuhn, Eds., Spec. Issue on Participatory Design, ***Communications of the ACM34***, 4, 25-26
- Nelson, D. (1993) 'Aspects of Participatory Design', ***Communications of the ACM34***, 10, 17-18
- Nelson, E. (2002) 'Extreme Programming vs. Interaction Design', January 15, 2002, ***FTP Online Magazine***: http://www.fawcette.com/interviews/beck_cooper/
- Norman, D.A. (1990) ***The Design of Everyday Thing***s, Basic Books, Doubleday
- Preece, J., Rogers, Y. and Sharp, H. (2002) ***Interaction Design: Beyond Human-Computer Interaction***, Wiley
- Scarbrough, H. and Corbett, J.M. (1991) ***Technology and Organisation: Power, Meaning and Design***, Routledge

- Turner, J.A. (1987) 'Understanding The Elements Of Systems Design', in Boland, R.J. and Hirschheim, R.A., Eds., ***Critical Issues In Information Systems Research***, Wiley.
- Visser, W. & Hoc J-M. (1990) 'Expert Software Design Strategies' ***in*** J.M. Hoc, T.R.G. Green, R. Samurçay, D.J. Gilmore, Eds., ***Psychology of Programming,*** Academic Press
- Winograd, T.A. (1994) 'Designing a language for interactions', ***interactions 1***, 2, 7-9
- Winograd, T.A. (1996) 'Introduction', T.A. Winograd, Ed., ***Bringing Design To Software***, ACM Press, v - ix.
- Winograd, T.A. & Flores, F. (1986) Understanding Computers And Cognition, Ablex Corporation, Norwood NJ

Page last updated 01/12/2007, Susan Gasson (sgasson@drexel.edu) ;