Introduction to the Special Issue of CSCW on Activity Theory and the Practice of Design

David Redmiles

Motivation from a Software Development Perspective

Design is an elusive concept! Herbert Simon called design an "ill-defined" problem [Simon 81]. Horst Rittel called it a "wicked problem" [Rittel 84]. Fred Brooks classified design as an "essential problem," which ordinary technological advances could not alleviate; only "great designers" could produce "great designs" [Brooks 87]. Design has always been an obstacle to software development projects. One reason design remains elusive is that the kinds of objects being designed continue to change. With respect to software applications, designers choose increasingly complex applications. The majority of computing applications has long since migrated from small, encapsulated codes of mathematical algorithms to distributed, multi-user systems.

Corresponding to the shift in applications, the definition of the complexity of a software design has shifted. Traditionally, complexity referred to an algorithmic measure: e.g., how many computations an algorithm could perform in a certain measure of time. More recently, the definition of complexity expanded to include the number of components and levels of abstraction comprising a software application's architecture. Today, and in this special issue, complexity is measured in terms of the activities in which a community of stakeholders are involved and in which software fulfills some purpose. If software projects are to succeed in real-world settings, then their proponents need to consider this expanded view of complexity.

To better understand complexity from this perspective, some software developers have turned to theories of human behavior at the individual, group, and organizational levels, and beyond [Grudin 94]. Activity theory (AT) is one such theory that seems particularly well suited to the endeavor. Activity theory provides a framework for describing phenomena at various levels. First, it answers software requirements questions at the most basic level, i.e., the tasks and activities the software is a part of. Second, it focuses on the social organization of key players in an activity, such as stakeholders in a problem, communities of users, roles, and other social forms. Many software projects have failed when they have missed or misidentified requirements having to do with these elements of activity theory. The papers in this special issue explore the richness of activity theory that will inform any researcher or practitioner seeking to understand critical factors affecting software that facilitates collaborative work. Nine papers are contained in this special issue followed by an independent commentary and a brief response to the commentary.

Specific Papers

The special issue begins with a paper by Barthelmess and Anderson, which nicely achieves two goals. First, it provides a condensed history of software engineering, highlighting obstacles, especially those having to do with collaborative work. Second, it explains some key elements of activity theory, making very clear comparisons between software engineering concepts and activity theoretic concepts. The topic of Barthelmess and Anderson's research is of special interest to software engineers: process-centered software development environments (PCSDEs). Their activity theory framework focuses on three levels of activity: coordinated, cooperative, and coconstructive levels. They identify the strengths and weaknesses of PCSDEs at each of these three levels. The greatest degree of deficiency is found at the coconstructive phase. The tools are strongest in their ability to facilitate individual work (coordinated level) and to some extent to accomplish team objectives (cooperative level). The tools are weakest for reflecting on the process representation for future improvements to the collaborative effort (coconstructive level).

Their analysis shows the difficulties that any software system must overcome to capture essential elements of group activity.

The paper by Clases and Wehner provides models for an AT analysis of problems surrounding knowledge management software for cooperative environments. They emphasize the issues of *multiple perspectives* and *unexpected events* in collaborative work. Multiple perspectives naturally arise when multiple communities of practice are represented in cooperative work. Unexpected events become opportunities for different communities to recognize differences in perspectives and reconcile them.

The issue of multiple perspectives is especially significant for knowledge management systems. In particular, knowledge management systems attempt to support cooperative work by making the right information available at the right time [Fischer 98]. However the problem contexts in which knowledge is entered is different from the context in which it is retrieved. Moreover, the very activity of entering knowledge is different from retrieval. Finally, the person entering the knowledge and the person retrieving the knowledge will in general come from different communities of practice, have different roles, different histories, and different familiar mediating means.

Knowledge management systems generally provide little support for assisting in the transfer of knowledge, or the bridge between the two activities of storage and retrieval, the contexts of problems at the time of storage versus retrieval, or the backgrounds of the authors and consumers of the knowledge. Thus, human actors are still needed to mediate the transfer and application of knowledge.

Clases and Wehner develop their design considerations for software systems for CSCW by studying a group that supports procurement, administration, and maintenance of software in an organization. The group is concerned with knowledge management in the form of project documentation. Clases and Wehner utilize a data collection and analysis technique called the repertory grid. The approach elicits concerns and issues identified with individuals' perspectives. Clases and Wehner propose a model of *corrective cooperation* in which multiple perspectives and unexpected events (such as breakdowns in communication) are anticipated. In the model, breakdowns or unexpected events transform cooperation into co-construction, thus expanding the cooperation. They conclude that "computer support for knowledge management practices should rather help to 'manage' than to reduce differences between actors' perspectives and locally evolving work practices."

The themes of knowledge management, tensions and breakdowns, and the need for human actors to mediate activity continue in the paper by Collins, Shukla, and Redmiles. They report on a case study of the development of requirements for an evolving knowledge management software system in a customer support group within a large organization. The requirements were identified through open-ended interviews with representatives of various communities of stakeholders: customer support engineers, activity theory researchers (the authors), corporate managers, and software designers. Engeström's activity theory model and his refinement of Wartofsky's mediating means hierarchy are used to structure the collection and analysis of the interview data. These models also served as the basis for interventions to improve current work practices within the different communities of stakeholders—i.e., to evolve the activity systems ,and associated mediating means, being studied.

Three relevant activity systems became apparent during the investigation: knowledge authoring, customer support, and knowledge maintenance. The paper focuses primarily on the activity system of knowledge authoring since the purpose of the investigation was to develop requirements to evolve the software system currently supporting this activity. However, new requirements were revealed by analyzing the tensions between the three activity systems. Very briefly summarized, the short-term objectives and organizational motivations for customer support were in conflict with activities of knowledge authoring and maintenance. Tensions within and between elements of the single activity system of knowledge authoring were also found. Some of these include uncertainty of roles, lack of awareness in a geographically distributed community, and complexity of intended mediating means. These tensions highlighted areas where direct improvements to the existing software were needed.

An unusual aspect of the research was the intentional and unintentional intervention that led to the improvement of current work practices. The initial interviews with stakeholders improved their awareness of other members of the organization relevant to their work as well as improving their knowledge about mediating means at their disposal. A briefing by the researchers improved the understanding between knowledge producers and consumers.

The paper concludes with extensive comments on the difficulties and the benefits of using activity theory in software design. Difficulties arose when aspects of the activity theory models had to be explained to stakeholders. Benefit came from the structure that the activity system model provided for collecting and analyzing data and the identification of mediating means.

The theme of tensions within and between activity systems, as well as the necessity of carefully analyzing activity systems surrounding an intended software system recurs in the next paper by Spasser. The theme of process, so central to Barthelmess and Anderson's work, also recurs. Spasser reflects on the development of a web-based project coordination and publishing environment. The environment, referred to as CPS, facilitates the publishing of a variety of stylized information about the flora of North America. The development of CPS was not merely the development of a digital library, but a research effort into, and evolution of, a collaborative process for multiple communities of stakeholders such as project leaders, content providers, content editors, and technical support staff. A realist case study methodology was used prominently in the collection of the data from which the software environment and collaborative process evolved. An activity theory analysis, influenced by a realist approach, revealed specific conflicts and tensions between different communities of stakeholders, including conflicts about objects such as published manuscripts. The CPS software was also a source of disagreement as it embodied conflicting objects. The identified conflicts and contradictions drove the evolution of a new, distributed activity system that the different communities agreed to adopt. Thus, Spasser shows how activity theory can facilitate software development in contentious settings and provides a perspective on the realist approach in general.

Tensions and multiple communities of stakeholders are also examined by Korpela, Mursu, and Soriyan as they discuss an AT framework for software development. The framework is based on the authors' own previous research, Leont'ev's work on motives and outcomes, and Engeström's models. The authors illustrate how the framework can be applied to software development projects using a hypothetical example. They also discuss their evaluation of the framework in a group of Nigerian software companies. The framework seems especially well suited for software companies or groups within organizations working in a consulting relationship with end users. The evaluation revealed an important result: namely, that the software developers and end users found the framework easy to comprehend and apply. Comprehensibility of methods has always been a stumbling block to software engineering and thus this result is very promising.

The framework illuminates several issues. First, the framework addresses several activity theoretical properties of software development: outcome, object and process, actors, means of work, means of coordination and communication, group versus team effort, and mode of operation. Second, the framework carefully identifies the different communities that come together to carry out software development: software professionals, academics, management, end users, and end users' clients. Third, the framework reveals areas where tension or conflict arises in software development, especially between different communities of stakeholders.

With respect to tensions in software development as an activity, the authors elaborate on the notion that software development is typically an activity that takes place along the boundary between communities. The two major communities are the software developers and the end users. Both generally have different histories and thus draw on different sets of experiences when trying to communicate and collaborate on a single project. There is also an inherent tension in a "temporary gathering" such as a software project team – learning to communicate for a short problem solving session without any promise of future collaboration.

In general, the authors make a call to the community for more empirical methods in software development. They contribute to this goal by providing a framework to support further empirical work, both on methods of software development, as well as in practical software development application situations.

Tensions arise as a source of software requirements and a catalyst for the evolution of activity systems in a paper by Miettinen and Hasu. They study the activity of innovation and, in particular, the transition of a medical instrument called a neuromagnetometer from a research prototype to a product in clinical use. The authors describe an analysis scheme with the following three levels: the use value of the product, the development of complementary means—a concept similar to mediating means, and the situated use of the product. Five communities of stakeholders are identified, three producers of the technology and two groups of consumers. For the purposes of collecting data from different stakeholders, a questionnaire was developed that focused on two levels of the authors' model: the use value and the situated use of the product. The questionnaire was used in conjunction with a workshop to collect data about the activities and viewpoints of different stakeholders. All stakeholder communities were represented at the workshop. The results highlighted tensions and conflicts arising from cultural-historical differences in the communities. It also revealed details of different activity systems. The authors' data provided a rich source of requirements.

There are several unique aspects to this work. It provides detailed discussions of innovation as an activity and the interplay between the innovation activity systems and producer-consumer activity systems. The work identifies tensions, conflicts, and contradictions as sources of requirements for products as well as catalysts for the evolution of activity systems. The multi-level analysis and specific questionnaire that proved useful in this case study are directly applicable to other studies of networks of activity systems. The particular use of the questionnaire in conjunction with a multi-community workshop reiterates the usefulness of activity theory as a guide to intervening in communities in order to better develop product requirements and evolve networks activity systems. This kind of intervention is more theoretically grounded than the typical approaches to joint design methods, including participatory design. Activity theory places a systematic, formal emphasis on identifying roles, actors, and communities relevant to a project and, very critically, an understanding of conflicts and tensions in the intended activity early in the design process. Thus, it allows collaborative gatherings between communities of stakeholders to be particularly fruitful with respect to providing information for product development.

The earlier in the development process that critical requirements can be identified, the more effective and less expensive a development effort will be. This result has been known for many years in the software engineering community. As a consequence, great emphasis has been placed on research in requirements engineering and process, the process of software development as well as the process surrounding the use of software products in the end users' work context. The authors illuminate effective methods for both of these areas of research.

Another challenging concept is that of software evolution. Software engineering research in regards to evolution tends to focus on specification languages (e.g., architectural description languages—ADLs), designs (e.g., component-based design), and tools (e.g., configuration management). However, another perspective is suggested by this paper, that of understanding the evolution of activities surrounding the innovation and introduction of products.

The next two papers present different aspects of virtual and augmented realities made possible by computing. Fjeld, Lauche, Bichsel, Voorhorst, Krueger, and Rautergerg describe the design and iterative improvement of an augmented reality hardware and software system called BUILD-IT. Their software allows groups to plan designs for any kind of physical system. BUILD-IT allows its users to work at a table, moving and placing physical blocks on the surface. All the while, a software system monitors the activity, and augments the experience with various visual projections. For example, if a block is supposed to represent a house for a group planning a neighborhood, then once placed, the block leaves the image of a house projected onto the table, and the block may be moved to another location to represent and facilitate the placement of another object. The software and hardware implementing this system are very sophisticated. However, the subtle points of the theory behind the system are equally impressive.

Fjeld and colleagues use BUILD-IT to research the interrelationship of social activity and creativity and how the creative, cognitive processes of internalization and externalization play out in a computer augmented reality. They rely heavily on Leont'ev and Engeström's theories of tools and exteriorization. Work or play in the social context stimulates creative thought internally. Internal thought becomes instantiated or externalized in physical tools (or other mediating means). These mediating means support not only an individual's reflective thought, but also support the communication across different communities of stakeholders. The theory of externalization is reminiscent of the theory of reflection-in-action by Donald Schön [Schön 83].

A further consideration for design is the emphasis on "handling aspects" and "fluidity" of mediating means espoused by Bødker, Kaptelinin, and others. This emphasis led to a variety of design experimentation with the bricks used to symbolize design components as well as with the virtual reality, projected images that reflected the symbolic actions of placing and manipulating the bricks.

Of interest as well are two figures in the beginning of the paper that contrast the distinction between planning as an exploratory process without precise goals vs. the traditional goal-oriented planning from earlier work on artificial intelligence and cognitive science. The authors conclude with some design guidelines based on their iterative evaluation and improvement of the BUILD-IT system. The guidelines are linked to their interpretation of the theories of activity mentioned above.

Zager also studies software from the perspective of a virtual world. He identifies a new kind of collaborative entity called a *coalition,* which is distinguished by the notion that the virtual world of a software system creates a *pseudo-collective object.* Zager's domain of study is that of distributed software applications for brokerage information. He reports on a software system called Trinity that maintains a virtual world of information about the distributed applications and underlying network that supports them. Trinity maintains information from many perspectives, relevant to each community of stakeholders. When a problem or breakdown in the network arises, symptoms of the breakdown are apparent in the different perspectives and different stakeholders can react accordingly. Network administrators can communicate to applications managers the implication that a failed network card could keep a particular application from running. The network administrators or the applications managers can communicate the implication that users may have trouble starting certain applications until the problem is fixed. In a coalition, stakeholders are relatively independent of one another until a problem arises. They detect the problem through the perspectives maintained by the virtual world view and react accordingly. There is no one tangible object coordinating a collaborative activity, but rather the virtual world consisting of multiple perspectives serving as what Zager calls a pseudo-collective object. Thus, coalitions arise without the usual framework of a collaborative activity: "What differentiates the coalition from the organized collaborative pattern is that the organized pattern is a mediated activity (i.e., it has a collective object) while the coalition is not [a mediated activity]." Zager provides a set of requirements for systems to facilitate coalitions. The requirements emphasize different aspects of maintaining multiple perspectives surrounding a problem so that stakeholders can appropriately be aware of a problem and share information to solve it.

One of the major strong points of the coalition concept is that it allows for the analysis of unplanned activities among many different stakeholders. Zager identifies many stakeholders involved in a brokerage firm's activity. However, he also makes it clear that there are many more potential stakeholders depending on the task at hand. The concept of coalition seems well suited to a more ad-hoc model of group activity as opposed to the more traditional emphasis on structured collaborations popular among designers of software systems.

The paper section of the special issue concludes with a contribution by Nardi, Whittaker, and Schwarz. They report on a field study of how professionals carry out work in today's interconnected world, and identify an alternative concept complementary to Zager's coalitions, i.e., intensional networks, i.e., ego-centered personal social networks. Because of modern telecommunications and computing infrastructures, including the Internet, it is possible for companies and individuals to work in a distributed fashion, both geographically and organizationally. Work that could previously be accomplished only by having workers co-located and employed for long careers for one company and at one site is now often accomplished by orchestrating diverse people intensional networks. Individuals employed fulltime by a company or hired on a temporary basis organize themselves to solve specific, short-term problems. This approach is attractive to businesses because it saves the overhead of having fulltime employees during times when no project is available to use their expertise. The approach is attractive to those individuals who seek more independence in the workplace.

However, there are hidden costs. More specifically, there is work to be done to enable this business model to work, work that the authors aptly note that does not appear in a business "process" chart. The authors term this work, netWork. Individuals perform netWorking to build and maintain personal relationships with other individuals who can assist them in completing work tasks. Thus, in this research, a network consists of living nodes with interconnecting links based on personal and professional relationships. Specific nodes are activated when they are applicable to specific tasks. The authors note how individuals can become more important than teams in collaborative work. There are two senses in which this new focus is true. The network consists of individuals and a single individual can be a point of focus for accomplishing a task.

The authors draw heavily on in-depth interviews to elucidate the issues making netWorking critical, difficult, and rewarding in today's workplace. Issues related to activity theoretical notions of mediating means, history, tensions, and conflicts play a critical role. Tensions arise in the need to bring together teams that normally do not work together. Individuals have different histories in their communities, different rules for activities.

The authors relate their work to Engeström's notion of knotworking, Wenger's research on communities of practice, and, Zager's analysis of coalitions. The distinguishing factors of netWorking center on the scope, specificity, and duration of the relationships, the establishment of roles, the duration of the activities, and motivations. All of the issues the authors identify present requirements that software designers must address if they are to support collaborative software that adheres to modern approaches to work. The work is very relevant, for example, to software companies that outsource development of code.

Common Themes and Threads

The papers weave a framework of design based on activity theory. Several themes recur, and thus deserve particular notice. In order to follow the description of the themes, the reader is directed to the following table:

Paper	**Authors**
1	Barthelmess and Anderson
2	Clases and Wehner
3	Collins, Shukla, and Redmiles
4	Spasser
5	Korpela, Mursu, Soriyan
6	Miettinen and Hasu
7	Fjeld, Lauche, Bichsel, Voorhorst, Krueger, and Rautergerg
8	Zager
9	Nardi, Whittaker, Schwarz

All of the papers study **collaboration** in one form or another, either the **collaborative design** of software (1, 2, 3, and 5) or more general products (6 and 9), or, **collaboration through the software** system (1, 2, 3, 4, 7, and 8). Some of the papers study the particular issues surrounding **knowledge management** systems in collaborative environments (2 and 3). Another specialty within collaboration is that of **virtual or computer-augmented realities** (7 and 8). The theme of **process** in collaboration is highlighted in two papers (1 and 4).

All of the papers are informative to **software design** and field analysis by providing implicit or explicit guidelines and case studies. However, several of the papers provide explicit **checklists** or **methods** of analysis (2, 5, 6, and 7).

Almost all of the papers comment explicitly on **mediating means** component of activity theory. Some emphasize the importance of the **history** of mediating means and other concepts in different communities of practice (1, 2, 4, 5, 6, and 9). Others provide extensive **examples** (9) or both examples and **classification** (3). The two papers on virtual and computer-augmented realities (7, 8) provide the most novel views of mediating means, with one (8) providing a new **definition.** Two papers (2 and 3) emphasize the on-going need to define **human actors as mediating means**.

All of the papers clearly identify several **communities of stakeholders** either in the collaborative design of software systems or the collaboration through software systems. Some emphasize the criticality of the **boundary** between different communities (2 and 5). Different communities of stakeholders give rise to **multiple perspectives**. Several of the papers emphasize the need to accommodate multiple perspectives either in the activity of design (4, 6) or problem solving (2, 8) or in the software systems that facilitate activity (2, 8). The discussion of stakeholders usual coincides with a discussion of the **history** of mediating means within those communities (1, 2, 4, 5, 6, and 9) noted above.

Tensions, conflicts, and contradictions of and between activity systems are a recurring theme in many of the papers. Differences between **communities of stakeholders** give rise to some tensions, especially when the communities **historically** have different mediating means (2, 3, 6, and 9). Differences between the context of input and use are identified as a source of tension in one **knowledge management system** (2). In the other knowledge management system (3), tension arises in the conflict of knowledge input and other activity systems. In the **virtual and computer-augmented reality systems** (7 and 8), the systems ameliorate tensions. Tensions and breakdowns serve as a driving force for the formation of **teams** (8 and also, to some degree, 7 and 9). Tensions, conflicts, and contradictions serve as a source of requirements for **software design** (2, 3, 6, and 4) and as the motivation for **evolution** of activity systems (1, 2, and 6). The interplay between these last two concepts should not be overlooked. Few software developers to date have anticipated the evolution of activities affecting their software. Finally, one is left with the sense that tensions, conflicts, and contradictions, central in most activity theory analyses, provide insight into human creativity.

Two of the papers (3 and 9) make explicit mention that activity theory guided the authors' **intervention** in the activity systems under study. From an experimental perspective, intervention is to be avoided. However, in the interest of adoption of software systems, intervention is often necessary to adoption. That activity theory can guide intervention to improve adoption and requirements gathering is a useful insight and suggests a more integrated view of software development, namely one that designs both the software system and the environments into which the software is deployed.

Conclusion

This special issue is a rich source of information about activity theory and case studies surrounding software design. It is informative to the collaborative design of software as well as the design of software for collaborative settings. If software projects are to succeed in real-world settings, the whole view of software needs to be taken in mind, from the stakeholders involved in the setting to the activities that the software is being used for, to the other activities that will surround and conflict with the software application. Many researchers and practitioners strive for a more theoretically-driven as well as empirically-driven software development process. This special issue is a further step in that direction.

References

[Brooks 87] F.P. Brooks, Jr. *No Silver Bullet; Essence and Accidents of Software Engineering,* IEEE Computer, V. 20, No. 4, April 1987, p.10-19.

[Fischer 98] G. Fischer. *Seeding, Evolutionary Growth and Reseeding: Constructing, Capturing and Evolving Knowledge in Domain-Oriented Design Environments,* Automated Software Engineering, V. 4, No. 4, October 1998, pp. 447-464.

[Grudin 94] J. Grudin. *CSCW: History and Focus,* IEEE Computer, V. 27, No. 5, May 1994, pp. 19-27.

[Rittel 84] H.W.J. Rittel. *Second-Generation Design Methods,* in N. Cross (ed.), Developments in Design Methodology, John Wiley & Sons, New York, 1984, pp. 317-327.

[Schön 83] D.A. Schön. *The Reflective Practitioner: How Professionals Think in Action,* Basic Books, New York, 1983.

[Simon 81] H.A. Simon. *The Sciences of the Artificial,* The MIT Press, Cambridge, MA, 1981.

Acknowledgements

The special issue has been the result of many people's efforts. First, we (Nardi and Redmiles) would like to thank the authors for their work and diligence to make their contributions and see them through to fruition. Second, we would like to thank the reviewers. Third we would like to thank the Editor-in-Chief, Dr. Kjeld Schmidt, for supporting this special issue.