Presented at AMCIS 2003

Now the Twain Shall Meet: Combining Social Sciences and Software Engineering to Support Development of Emergent Systems

Sandeep Purao  
School of Information Sciences and Technology  
The Pennsylvania State University, University Park  
State College, PA 16803

Duane Truex  
Chapman Graduate School College of Business  
Florida International University  
Miami, Florida 33199

Lan Cao  
Department of Computer Information Systems  
Robinson College of Business, Georgia State University  
Atlanta, GA 30345
1. Introduction

Much current research in information system development (ISD) focuses on creating IT artifacts (Information Systems) that support users’ activities in organizational contexts. The software engineering research community, primarily responsible for techniques and processes for developing these artifacts, has paid insufficient attention to the complex relationship between the Information System (IS) and the organizational context in which it is embedded. On the other hand, branches of the MIS community, such as the IFIP Working Group 8.2, have paid limited attention to translating their insights on organizational context into actionable techniques that may be used for building more effective information systems. Without the benefit of integration, these two streams have traveled down paths that may be viewed as increasingly specialized, making dialog difficult between the two communities. We suspect that this lack of dialog has resulted in increasing emphasis on minutia and decreasing relevance of the research outcomes to the IS community. As an applied research discipline, IS research is concerned with the understanding of IT in organizations and society. Without the benefit of an integrated perspective, the research outcomes, therefore, are likely to touch upon only parts of the solutions required for solving complex problems.

This paper makes an attempt to combine important insights from these two research streams. A specific goal of the paper is to argue for the development of specific representational techniques that may help achieve this integration. The remainder of the paper is organized in three sections. Section 2 outlines theories that have driven investigation of IT in organizations. In section 3, we describe current research thrusts in software engineering. Finally, section 4 develops the argument for specific techniques and models – particularly, representational techniques – that can help operationalize the lessons learned from social theories with the help of tools available to software engineers.
2. Social Science Theories to understand IS/IT in Organizations

Understanding the relationship between technology and society is at the heart of much this IS research tradition. Social theories try to understand, explain and predict social behavior. They have long been used to help explain the underlying logics of social organizations. In the IS Research discipline the IFIP WG 8.2 research community provides a microcosm of the social theory in IS debate. In its publications one can find a very extensive use of and debate about the efficacy of various social theories. In a recent analysis of social theory citations in the IFIP WG8.2 proceedings, Jones (2000) showed how the use of different social theories has evolved within the WG 8.2 community since its inception in 1977. We use this research community and Jones’s research as a surrogate of the whole of our research community to illustrate our point. He performed a content analysis on the working group proceedings identifying the predominant theories each paper in those proceedings. The following table illustrates relative frequencies of the major social theories employed in these works.

<table>
<thead>
<tr>
<th>Social Theorists</th>
<th>77-99</th>
<th>92-99</th>
</tr>
</thead>
<tbody>
<tr>
<td>Giddens</td>
<td>50</td>
<td>60</td>
</tr>
<tr>
<td>Habermas</td>
<td>60</td>
<td>40</td>
</tr>
<tr>
<td>Berger</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>ANT</td>
<td>40</td>
<td>50</td>
</tr>
</tbody>
</table>

Table 1: Social Theorists cited in IFIP WG 8.2 (Adapted from Jones, 2000).

One theoretical perspective that found particularly relevant to research in ISD is Giddens’ Structuration Theory, which can be used to capture the complex relationship between the IS and its organizational context. [Giddens, 1979, Giddens, 1984, Giddens, 1987, Giddens, 1990, Giddens and
While the application of structuration theory to IS research continues to be a debatable issue, most would agree that the theory is well suited for exploring the ephemeral boundaries between IT and organization. Several authors have used it to explore how IT’s dual character and enables and constrains social action. Thus adaptations of structuration theory have proved useful to study various aspects of information technology in organizations [Barley, Barley, 1986b]; [Rose and Lewis, 2000]; [Orlikowski and Robey, 1991, Orlikowski, 1992]; [DeSanctis and Poole, 1994, Walsham, 2002, Walsham and Han, 1991]. A key principle in structuration theory is that of the duality of structure: human action is enabled and constrained by structure, but structure is also the result of human action. Thus, the duality in structuration theory refers to the way in which action and structure presupposes each other. That is, social structure provides enabling and constraining elements that are drawn on in human interactions, and in so doing social structures are produced and reproduced.

3. A Software Engineering Perspective on IS/IT in Organizations

As the name suggests, the software engineering perspective considers IS/IT as an artifact to be ‘engineered,’ that is, developed, built, and deployed. The primary focus, therefore, is on techniques and models that will allow design and production of software artifacts. The organization, then, is often seen as the source of requirements, which must be converted to formats appropriate for realization in the software artifact. Much of the ISD literature, therefore, considers the IS artifact as an enabler (e.g., Davenport 1993) or driver to trigger previously unplanned organizational shifts (Davenport 1993; Nance 1996; Robey and Azevedo 1994) or as the embodiment of the structure and rules of the organization (DeSanctis and Poole 1994). These works take a dualistic approach, in seeing the two domains of the IS artifact and the organization as being separate rather than a dualism in which the two are mutually joined in a dance of perpetual structuring, each influencing the other – a perspective repeatedly put forward by the social theorists. Software engineering research has, therefore, paid limited attention to this interaction between IS and organizations. Many methods and the concomitant modeling techniques assume relative stability
in organizational problem domains and proceed to quickly move to developing generic organizational use case, or process instances.

Research within ISD does, however, suggest that the problem domain (organizational context) is intertwined with the design domain (the IS artifact), and may change with the opportunity or need to redesign the business process (Mathiassen and Stage 1992; Welke 1994). Bergman et al. (2001), for example, point out that ISD process creates goals that require adoption of both the business process as well as design of the information system. This interplay is captured by the ideas of ‘Problem’ and ‘Design’ spaces, which represent a key dimension that underlies ISD processes (Guindon 1990; Mathiassen and A. Munk-Madsen 2000; Purao 2002). Purao et al (2002) employ grounded theory development techniques to present evidence of the existence of these two spaces, and describe patterns of behaviors across the two spaces. Their study, accordingly, laments the lack of supporting techniques to bridge the gap between problem and design spaces – mirroring the larger concern voiced by Mathiassen and Stage (1992) and Bergman et al. (2001) and suggesting possible reasons for documented problems with the failures in Information Systems development and acceptance (Dardenne et al. 1993).

In spite of this recognition of interplay and dynamics across the two spaces, a key goal of current ISD practice and research has been the minimization of change, either with better articulation of requirements or flexible design, i.e. assuming either an unchanging problem space or adjusting the design space in response to changes in the problem space. Accordingly, several modeling techniques have been proposed to ‘deal with complexity,’ ‘allow for flexibility,’ and ‘facilitate communication between developers and users.’ As a result, several modeling and representation techniques have been developed to capture different perspectives of the underlying IT artifact – including functionality, structure, behavior, and dynamics. Modeling techniques for ISD have primarily dealt with the design space, with the exception of a few, informal techniques that have addressed the problem space. Developing an information system, however, requires understanding of problem space – e.g. domain knowledge which is considered informal, implicit, ad hoc, and modeled only incompletely and
indirectly in terms of problem-specific languages (Iscoe, Williams et al. 1991). A few techniques have, in fact, been proposed to model the problem domain such as the use of flowcharts and dataflow diagrams to model the current functions (Gane and Sarson 1979), the IDEF suite of process maps (Mayer, Benjamin et al. 1995). These have found little acceptance in practice, and have generally been considered too complicated for users to understand (Dawson and Swatman 1999). Further, they have not directly addressed concepts such as organizational actors, activities and tasks; instead, focusing on only those aspects of the organizational context that will eventually be represented in an information system.

Another technique is rich pictures (Avison and Wood-Harper 1990), which are cartoon-like representations that identifies all the stakeholders, their concerns, and some of the structure underlying the work context. This technique, while easy to use, has suffered from lack of integration with more formalized approaches.

On the other hand, in design space a large number of methods have been proposed for the specification of software systems in the past 20 years. Until 1988, the proposed methods followed the structured approach and since that time, most proposals have followed the object-oriented approach. A survey by Wieringa (Wieringa 1998) classified the techniques as those modeling the functional, behavioral, and communicational properties of a system as well as decomposition techniques. Examples of these techniques include: function refinement trees, event-response specification, uses cases, process graph, process structure diagram, state transition diagram, state charts, Data Flow Diagrams (DFD), sequence diagrams, collaboration diagrams, context diagrams, Entity-Relationship Diagrams (ERD) and class diagrams. In terms of the framing proposed by Wand and Weber (2002), these techniques have concentrated on metaphors such as accurate representation, richness and variety of representations, appropriate syntax, grammar and script. There are few techniques, if any, that have dealt with the important concern of representing context, and further, influencing context. These modeling techniques have, therefore, continued to focus on the design space (see Figure 1).
4. ISD as Co-Evolution of Problem and Design Space

Both streams have been slow at recognizing that the uni-dimensional views embraced within the stream may be inadequate to deal with the difficult problems associated with developing complex information systems. ISD process is a dynamic process that needs to capture the changing requirements and constraints. Change management is a fundamental activity in requirement engineering (Nuseibeh 2000). To deal with the interaction with the changing business environment, prior research in ISD has focused on various process models. For example, spiral model (Boehm 1988) describes software development as the iteration over four phases of activity. The specification and plan are refined at each iteration. Another mechanism to manage the changing environment is to trace changes to requirements, design and implementation of the system. With traceability, requirements are linked to their sources and to the artifacts created during the ISD cycle. Therefore, the impact of change in requirements can be identified (Ramesh and Jarke 2001). Bergman et al (2001) have suggested that requirements analysis should be seen as an iterative process from an existing solution space to problem space, then to the future solution space. While their characterization of problem and solution space is somewhat different from the characterization described in the sidebar 1, their idea of heterogeneous engineering does bring out anomalies in existing business process not supported by the current systems as the problem space. Their
metaphor of ‘grinding’ to capture this dynamic nature of problems and solutions, and their continuous discovery and simultaneous interpretation/creation (see Figure 2) is, however, clearly relevant to our characterization of problem and design spaces.

Figure 2. Heterogeneous Engineering [Adapted from Bergman et al. [2001]]

As the discussion above suggests, the software engineering stream is beginning to realize that the exclusive focus on the design space (the IT artifact) overlooks the possible intervention that a system developer may be able to make in the problem space (the organization). Problem space and design space represent two distinct arenas of influence for ISD. The problem space is a subset of the real world that the computer-based information system is designed for. It describes the business processes, roles involved in the processes and distribution of power and resources. Problems of current system and user requirements of a system are analyzed based on understanding the problem space. The design space is the solution of the problems identified in the problem space and the design of the system itself. The idea has been alluded to and received attention over the years (Guindon 1986; Guindon 1990; Simon 1996; Mathiassen and A. Munk-Madsen 2000). Developer behaviors in the problem space differ considerably from those in the design space (Purao 2002). For example, in problem space, developers engage in concept simulation in an effort to explore different concepts and visualizing a mental representation of concepts. In design space developers engage in behaviors such as expanding mental models, proposing new concepts, considering action sequence among those new concepts and visualizing how the artifact may behave. Successfully engaging in each space is critical to the quality of the system. While there are tools and techniques supporting behaviors in design space, very few techniques or procedures are available in supporting developer behaviors in the problem space (Purao 2002). If the ISD process is recast as
simultaneously causing changes in both spaces, it can be seen as proactive in modeling, simulating and evaluating the changes and the effects of the changes on both the organization and the information system being developed [Purao and Cao 2003].

The social theories stream is also slowly shifting its agenda, borrowing from the theory of linguistic and organizational emergence, to the line or work loosely called “emergent systems development” or “deferred systems development.” [Bansler and Havn, 2002, Baskerville et al., 1996, Bello et al., 2002, Patel, 1999, Patel et al., 2002, Truex et al., 1999, Truex et al., 2000]. “Deferred system’s design (DSD) provides some elements to consider in designing context-aware IS. Most context-aware systems record information on users’ actions for later use. In IS this is reflected in work that seeks to predict users’ future requirements. The IS design is based on predicting what users will require in order to implement adaptation. The DSD approach does not seek to predict specific future uses of an IS or develop designs on a predictable basis. It proposes …design principles that can be used to develop a basic research agenda in deferred design. …Developers would need to build these into deferred systems and enable users to match their environment or context to the system through tailorable software. 

A deferred system is deferred until the user decides what the system will become. In philosophical terms, this provides for ontological and epistemological relevance of the system to users’ being. In business terms, it caters for the situated needs of system users.” Emergent systems development views organizations as emergent and then illustrates how there should be a consequent shift from traditional development practices to a "continuous redevelopment" process. [Truex et al., 1999]

5. A Research Agenda

In general, techniques are needed to facilitate this move – we focus this discussion on representational techniques -

Representational techniques are needed to bridge problem and design spaces. A representational technique is important because it brings together a community of researchers by focusing attention on key aspects that can be abstracted in the form of modeling constructs. It allows the research community to
come to an agreement about key constructs within a discipline. A good example of how a research community can get organized in this manner is the entity-relationship diagramming technique proposed by Chen (1976). The simple yet elegant proposal allowed the research community to begin thinking about research in databases in the form of a metaphor that captured the essence of databases as reflecting a universe of IS course. More recently, the agenda by Wand and Weber (2002) has presented such an opportunity. Our approach represents a call for extending that agenda to include the problem space.

The problem of developing techniques for modeling the problem space, and for bridging models across the two spaces is a difficult one. One of the difficult problems in modeling the problem space is choice of an adequate representation. The information obtained in problem space, particularly at the early stages, is usually informal, subject to rapid change, contain different personal views, demonstrate little system understanding, and be completely informal (Jarke and Pohl 1993). Developers then move to semi-formal graphics, and in some situations also to formal specifications. (Lubars et al. 1993).

To support such behaviors at different stages of a project, we may need a two-layer approach to represent problem space: the first layer is a textual representation layer that retains the basic functionality of a standard text-processing user interface while adding traceability to constructs in the second layer. The second layer—a graphical model layer is a more structured representation of the problem space.

References


Guindon, R., H. Krasner, and B. Curtis, (1986), Breakdowns and processes during the early activities of software design by professionals.


Mathiassen, L. and J. Stage (1992), "The principle of limited reduction in software design," Information Technology and People, 6 (2-3).


