Integrating Soft Systems Methodology into the Teaching of Human-Computer Interaction: A Constructivist Design Based on Problem-Based Learning

Kam Hou VAT
Faculty of Science & Technology
University of Macau, Macau
Macau SAR, China
fstkhv@umac.mo

Abstract: This paper describes the initiative and philosophy of introducing soft systems methodology (SSM) into the curriculum of human-computer interaction (HCI), enacted through the pedagogy of problem-based learning (PBL). Specifically, the idea of teaching scenario-based interaction design for complex problem solving is rendered as our educational response to the continual challenge in the field of HCI. Our discussion should serve as an experience report of an ongoing action research in the update of a curriculum that could fit in the undergraduate program of Software Engineering, Computer Science, or Information Systems. In particular, we depict some problem scenarios, helping the design of the course activities, and developing our students as self-directed work teams of software professionals. The paper concludes with the author's perceived challenges in the SSM-based HCI course enactment plus the necessary reflective evaluations therein.

Introduction

The teaching of human-computer interactions (HCI) (Vat, 2000, 2001) in the undergraduate curriculum has always been a challenge as it is composed of such a mix of elements as human factors, user expectations, man-machine interfaces construction, cognitive psychology, computer science, and those latest developments on contextual design in interactive systems. In the case of the author's teaching experience, since 1998, the pedagogy adopted to deliver such a course has been shifted from a conventional instructivist approach to the constructivist method of problem-based learning (PBL) (Greening, 2000; Ryan, 1993). Besides, with the increasingly accumulated course material to cover in a single semester, the idea of scenario-based design (Carroll, 2000) has also been incorporated in 2000 with an attempt to help undergraduate Software Engineering students deepen the idea that HCI is concerned with understanding, designing, evaluating and implementing interactive computer systems to match the needs of people. It is our experience that the constructivist's ideas of problem-based learning (PBL) (Barrows, 1986; Greening, 2000; Ryan, 1993) revolving around a focal problem, group work, feedback, skill development and iterative reporting, with the instructor playing the coach by the side, guiding, probing, and supporting student-groups' initiatives along the way, could help students develop a unified team-based approach to better manage the underlying software requirements. Methodically, we need some working scenarios to try out some iterative process involving researchers (instructor) and practitioners (students) acting together on a particular cycle of development activities, including problem diagnosis, action intervention, and reflective learning. Particularly, our action research approach should involve evaluating how well the students playing the role of practitioners, could function as self-directed work teams (SDWTs) of software professionals, following the constructivist's tenets of PBL, in performing software development for a specific user scenario. Against this backdrop, the use of soft systems methodology (SSM) (Checkland & Holwell, 1998; Checkland & Scholes, 1999) has demonstrated quite a promise in enhancing the student-practitioners' learning to deal with the design difficulties typified in the complex messy domain of ill-defined problem situations.

The PBL Paradigm of Collaboration

Problem-based learning, according to Barrows (1986), is designed to actively engage our students, divided in groups, in opportunities for knowledge seeking, for problem solving, and for the collaborating necessary for effective practice. At the heart of PBL is a set of group-based activities, including climate setting, starting a problem, following up the problem, and reflecting on the problem. A brief description of the PBL model of collaboration could be presented as follows:

- *The Climate Setting Phase.* At the outset, before the PBL group work begins, students must get to know one another, establish ground rules, and help create a comfortable climate for collaborative learning. Meeting in a small group for the first time, students typically introduce themselves, stressing their academic backgrounds to allow the facilitator (instructor) and each other to understand what expertise might potentially be distributed in the group. The most important task is to establish a non-judgmental climate in which students recognize and articulate what they know and what they do not know.

- *The Problem Initiation Phase.* The actual PBL episode begins by presenting a group of students with minimal information about a particular problem. Students then query the given materials to determine what information is available and what they still need to know and to learn to solve the problem. During this phase, students typically take on specific roles. An example is the scribe, who records the group's problem solving, including listing the facts known about the problem, students' ideas, additional questions about the problem, and the learning issues generated throughout ensuing discussion. Such written record helps the students keep track of their problem solving and provides a focus for negotiation and reflection. Throughout the problem-solving process, students are encouraged to pause to reflect on the data collected, generating additional questions about that data, and hypothesizing about the problem and about possible solutions. Early in the PBL process, the facilitator may question students to help them realize what they do not understand. As students become more experienced with the PBL method and take on more of the responsibility for identifying learning issues, the facilitator is able to fade this type of support, or scaffolding. After the group has developed its initial understanding of the problem, the students divide up and independently research the learning issues they have identified. The learning issues define the group's learning goals and help group-members work toward a set of shared objectives. These objectives can also help the facilitator to monitor the group's progress and to remind members when they are getting off course, or alternately, to ask if they need to revise their goals.

- *The Problem Follow-up Phase.* In the problem follow-up phase, students re-convene to share what they have learned, to re-consider their hypotheses, or to generate new hypotheses in light of their new learning. These further analyses, and accompanying ideas about solutions, allow students to apply their newly acquired knowledge to the problem. Students share what they have learned with the group as they interpret the problem through the lens of their newly accessed information. At this point, it is important for the students to evaluate their own information and that of the others in their group. In the PBL group, information is not often accepted at face value. Students must discuss how they acquired their information and critique their resources. This process is an important means of helping the students become self-directed learners.

- *The Problem Reflection Phasee.* During post-problem reflection, students deliberately reflect on the problem to abstract the lessons learned. They consider the connections between the current problem and previous problems, considering how this problem is similar to and different from other problems. This reflection allows them to make generalizations and to understand when this knowledge can be applied. Finally, as the students evaluate their own performance and that of their peers, they reflect on the effectiveness of their self-directed learning and their collaborative problem solving.

The Setup of PBL-Based Action Research

Throughout our course delivery, we have born in mind that our research should be evaluated in part by its ability to explain practice. Thereby, in each semester when our HCI course is offered (see (Vat, 2001) for more details), in order to help judge the research outcome with respect to the process of problem diagnosis, action intervention, and reflective learning, students embark on the PBL cycle of learning through organized groups of 4-6 members (one being the team leader). Each PBL group will be given a dual role to explore as client and as developer within a specified period of time. Namely, each team, acting as the developer, is to complete an interactive system design and prototype for another team acting as the client. Yet, the same team is the client of another group, responsible for clarifying the project, and resolving ambiguities as they arise, but in any pair of PBL teams (say, A and B), they cannot be the client and developer of each other at the same time. Meanwhile, each PBL team is required to present their work in progress, and lead class forums to elicit students' discussions. The team leader, equivalent to project manager, has to coordinate the team activities, and ensure effective team communications. And team members have to help set the project goals, accomplish tasks assigned, meet deadlines, attend team meetings and participate in editing project documents and integrating work-products to be combined as the final project report. At the end of each project milestones, each member of the respective PBL teams is required to make a presentation of his or her project involvement, with a question and answer session for the client team and the whole class. The instructor, acting as the project sponsor for each client team, and as the project supervisor for each developer team, designs the necessary scenarios to guide, motivate and provide feedback to the PBL groups. Also, the instructor has to evaluate how well students perform in the PBL groups, and how well such groups behave as SDWTs in managing software requirements, and provide the necessary adjustments following the ideas of soft systems methodology (SSM).

The Criteria of Managing SSM-Based Project Development

To partially address the management challenge in learning about HCI-related software development, a number of development criteria have been selected to enrich our PBL students' team skills experience in exercising soft systems methodology. These criteria are useful in evaluating the central task of systems development, which include: clarifying the problem situation, identifying design moves, envisioning the solution, recognizing trade-offs and dependencies, and anticipating impacts on human activity.

- *Clarifying the Problem Situation*. This first step in design problem solving begins with such questions as: What is wrong with the current state of affairs? What is needed? What could be improved? The standard approach in software development is to carry out some sort of requirements analysis. This analysis may initially be couched as a fairly high-level statement provided by the client – the person or organization that commissioned the design work. Such a statement may also be developed by, in collaboration with, or from observation of prospective users of the system to be developed; or it may be based on the hunches of the designers. Nonetheless, this initial requirements statement must be successively elaborated and refined to obtain a precise description of the situation that highlights the specific needs that the design work will address.

- *Identifying Design Moves*. To the extent that a design problem can be clarified, we need to move toward a solution. Typically, we do not know what specific moves are possible or useful a priori; part of the creativity of design is discovering the relevance and effectiveness of a move that has not been tried before. But this is obviously difficult. Much work on design methods has focused on describing what are sometimes called weak decomposition. The basic strategy is to organize an overall design problem into a set of component sub-problems, each simpler than the original problem. This process is re-iterated until the sub-problems are easily solvable, namely, as examples of known problems with known solutions. Nevertheless, starting design work with weak decomposition tends to simplify problems in ways that implicitly discourage creative solutions, bearing in mind that requirements typically change through the course of design work. Today, it is often experienced that an actively synthetic design method of planning by doing that is complementary to the analytic techniques of problem structuring and decomposition, is needed. Designers, nonetheless, might want to make provisional design moves within a concrete design space, explore and develop requirements, and test the consequences of such moves before committing to them.

- *Envisioning the Solution*. The objective in design is to specify a solution that satisfies the needs identified in the current situation. The design solution is typically described by such artifacts as: the technical drawings, diagrams and written specifications, which provide detailed guidance for those who will implement the design and for those who subsequently may debug, enhance, or otherwise maintain the designed solution. However, such specifications can be obstacles to the full participation in the design process of clients and prospective users, who speak the language of the use situation, but not the language of software specification often characterized by rendering the vivid and open-ended designs as stilted enumerations of features and functions. After all, the essence of an interactive system is that it is dynamic and responsive: How can this be merely captured in a static list of features and functions? Henry Dreyfuss, in his book *Designing for People* (Dreyfuss, 1955), energetically confronts these points. He wanted to present a design as something tangible, sharable with clients and prospective users; hence, he created a design paradigm of active, mutual engagement in which designers and their clients and users work in close coordination, noticing the world as it is and responding with mock-ups of the world as it might be.

- *Recognizing Trade-offs and Dependencies*. Creating a design solution involves subtle trade-offs and dependencies regarding functionality and usability. The sheer number of important details and their many interactions is an intriguing challenge of design. Often, structured design methods seek to manage interactions by grouping requirements and constraints to specify sub-solutions to sub-problems, and thereby to build up a comprehensive design solution. Understandably, the problem decomposition imposed through such methods shapes the ultimate solution, and may in fact conceal important trade-offs and dependencies. Stated another way, specifications that are developed strictly sub-problem by sub-problem cannot ensure an overall coherence in the design. Dreyfuss (1955) rendered a more concrete perspective on the issue of managing trade-offs and dependencies. He stressed the importance of empirical methods for instantiating and evaluating trade-offs and dependencies. These methods rely on the development of design mock-ups and observations of them in use. The understanding gained through these empirical means could then be used to refine the design solution.

- *Anticipating Impacts on Human Activity*. Designed artifacts have a myriad of consequences for people (Mirel, 2004) – some intended, some unintended, some that empower people and enrich their lives, and some that frustrate and punish people. They are complex agents of change; they alter our tasks and our social structures; they have both positive and negative effects, often at the same time and in virtue of one another. Historically, these complications work themselves out through trial and error. Doing better than this often requires sophisticated analysis of use situations coupled with flexible strategies to guide an iterative process of refinement and redesign. Typically, if we think of each design project as an isolated activity, we will not be able to see enough of the long-term consequences for people. However, few system designs are completely novel, and we do know some things about human activity and experience that appear to be relevant across many types of situations. Thus, in the experience of John Carroll (2000), there is the possibility of what might be called cumulative design, in which we observe the human impacts of past designs through time and attempt to direct that knowledge toward guiding the development of future designs.

Scenario-Based Design with SSM

Many of today's interactive systems (IS) are difficult to learn and awkward to use; they often change our activities in ways that we do not need or want. The problem lies in the IS development process. It has been observed that traditional textbook approaches to IS development seek to control the complexity and fluidity of design through techniques that filters the information considered, and weakly decompose the problems to be solved. In contrast, scenario-based design approach (Carroll, 2000) belong to a complementary tradition that seeks to exploit the complexity and fluidity of design by trying to learn more about the concrete elements of the problem situation. Thereby, John Carroll characterizes scenarios as concrete stories about use through which IS architects could envision and facilitate new ways of doing things and new things to do. Specifically, scenarios provide a vocabulary for coordinating such development tasks as: understanding people's needs, envisioning new activities and technologies, designing effective systems and software, and drawing general lessons from systems as they are developed and used. Namely, scenarios help IS designers analyze the various possibilities by focusing first on the human activities that need to be supported and allowing descriptions of those activities to drive the quest for correct problem requirements. It is expected that through maintaining a continuous focus on situations of and consequences for human work and activities, IS designers could become more informed of the problem domains, seeing usage situations from different perspectives, and managing trade-offs to reach usable and effective design outcomes.

Consequently, through the appropriate use of design scenarios, the challenge of designing IS support should never be thought of as something to be defined once and for all, and then implemented. Instead, it must be based on the observation that all real-world problem situations contain people interested in trying to take purposeful action (Checkland & Holwell, 1998; Checkland & Scholes, 1999; Mirel, 2004). Pragmatically, the idea of a set of activities linked together so that the whole, as an entity called the human activity system (HAS) from the viewpoint of Soft Systems Methodology (SSM) (Checkland & Scholes, 1999) could pursue a specific purpose, could indeed be considered as a representative organizational scenario for architecting IS support, which is never fixed once and for all. In practice, given a handful of the HAS models, namely, models of concepts of purposeful activity built from a declared point of view, we could create a coherent structure to debate about the problem situation and what might improve it.

Subsequently, from the IS architect's point of view, while conceiving the necessary IS support to serve the specific organizational service requirements, the fundamental ideas could be integrated as follows: Always start from a careful account of the purposeful activity to be served by the system. From that, work out what IS support is required (by people) to carry out the activity. Treat the creation of that support as a collaborative effort between technical experts and those who truly understand the purposeful action served. Meanwhile, ensure that both system creation and system development and use are treated as opportunities for continuous learning. In this way, models of purposeful human activities can be used as scenarios to initiate and structure sensible discussion about IS support for the people undertaking the real-world problem situations.

Designing HCI-Based Course Scenario

At the beginning of the semester, our course scenario begins when the instructor acting as project sponsor for each client team, and as project supervisor for each developer team, help the class evolve into its team-based organization, resulting in an even number of teams. It should be noted that an even number of teams is important because each team plays two roles, namely, one being the client, and the other being the developer. Each client team is handed a design project by the sponsor. It is then given some inception time to elaborate on the specifics of the project. At the end of the inception period, each client team is assigned a developer team from among the remaining client teams. When a developer team has been identified, the working and performance of the developer team is guided and monitored by the project supervisor played by the instructor.

In a typical semester, there might easily be six to ten teams of students, with each team composed of four to six members each. Essentially, each design project invites our PBL student-groups to embark on a journey to develop some interactive system that meets customers' real needs in Web-based development. The general requirement is for each PBL team to create and maintain a review Web site to keep all team members up-to-date on all possible aspects of the project. It is also where the PBL team will work (report) collaboratively on the project. Through the review Web-site, our PBL teams can conduct reviews with their clients, who can view their project in progress, give feedback on a design, get in touch with the PBL team, and check the project schedule. The review Web-site contains numerous information such as: the roles and responsibilities of the project team, contact information for all team members, the project mission, the vision document, the project schedule, and all design reviews.

It is designed that the first thing our PBL teams have to learn is a systematic approach to eliciting, organizing, and documenting the requirements of the system to be built for the client team. Also important is a process that establishes and maintains continuous agreement between the client and the developer teams on the changing requirements of the system. Individual PBL teams have to understand users' problems in their culture and their language and to build systems that meet their needs. Practically, the HCI context for the course is designed around four core development processes to be experienced by our PBL student-groups within the semester's duration constraint.

- *Analyzing the Problem*. This involves a set of skills to understand the problem to be solved before application development begins. It is the process of understanding real-world problems and user needs and proposing solutions to meet those needs. We consider a problem as the difference between things as perceived and things as derived (Gauss & Weinberg 1989). Accordingly, if the user perceives something as a problem, it is a real problem, and it is worthy of addressing.

- *Understanding User Needs.* Software teams are rarely given effective requirements specifications for the systems they are going to build. Often they have to go out and get the information they need to be successful. Typical methods include interviewing and questionnaires, requirements workshop, brainstorming and idea reduction, storyboarding, role playing, and prototyping. Each represents a proactive means of pushing knowledge of user needs forward and thereby converting fuzzy requirements to those that are better recognized.

- *Defining the System.* This describes the process by which the team converts an understanding of the problem and the users' needs to the initial definition of a system or application that will address those needs. Our PBL teams should learn that complex systems require adaptive strategies to organize information for requirements. This information could be expressed in terms of a hierarchy, starting with user needs, transitioning through feature sets, then into the more detailed software requirements.

- *Managing the Project Scope.* Project scope is presented as a combination of the functionality to be delivered to meet users' needs, the resources available for the project, and the time allowed in which to achieve the implementation. The purpose of scope management is to establish a high-level requirements baseline for the project. The team has to establish the rough level of effort required for each feature of the baseline, including risk estimation on whether implementing it will cause an adverse impact on the schedule.

Enacting SSM as a Learning Process

A very important activity for each developer team is to actively engage its client team in helping solve each of the four core processes described in the previous section to ensure the quality and timeliness of the software outcomes. Undeniably, setting up an interactive system (IS) is a social act in itself, requiring some kind of concerted action by many different people; and the operation of an IS entails such human phenomena as attributing meaning to manipulated data and making judgments about what constitutes a relevant category. In this regard, the use of scenarios in the creation of IS support, can be seen as a means which learns its way to the meanings which characterize an organizational context. This idea of learning the meanings, by which people sharing a human situation seek to make sense of it, is a significant feature of SSM.

The important point is that we must not lose sight of the fact that the HAS (human activity systems) models are not would-be descriptions of parts of the real world. Instead, they are abstract logical machines for pursuing a purpose, defined in terms of declared worldviews, which can generate insightful debate when set against actual would-be purposeful action in the real situation. The implicit belief behind constructing the HAS models is that social reality – what counts as facts about the social world inside any organizational context – is the ever changing outcome of a social process in which human beings continually negotiate and re-negotiate, and so construct with others their perceptions and interpretations of the world outside themselves, and the dynamic rules for coping with it.

Researching social reality in the context of IS development then becomes an organized discovery of how human agents make sense of their perceived worlds, and how those perceptions change over time and differ from one person or group to another. In the process, we do not expect to discover unchanging social laws to set alongside the laws of natural sciences. Rather, an organization is perceived as entailing readiness on the part of its members to conceptualize it and its internal and external relationships in a particular way, though it is also understood that such readiness changes through time, sometimes incrementally, sometimes in a revolutionary way, as perceptions and membership change.

The basic shape of the scenario-based learning approach could simply be described as follows: Find out about the problem situation that has provoked concern; Select relevant concepts that may be integrated into different human activity systems; Create HAS models from the relevant accounts of purposeful activity; Use the models to question the real-world situation in a comparison phase. The debate initiated by the comparison normally entails the findings of accommodations between conflicting interests, that is to say, situations that may not satisfy everyone, but could still be lived with, enabling action to be taken. Oftentimes, the purpose of the debate is to collectively learn a way to possible changes (improvements) to the problem situations, by activating in the people involved, a learning cycle, which counts on their ability to articulate problems, to engage in collaboration, to appreciate multiple perspectives, to evaluate and to actively use their knowledge. It is worthwhile to notice that taking the purposeful action would itself change the situation, so that the whole cycle could begin again, and is in principle never ending.
Likewise, through scenarios, IS architects could provide help in articulating the requirements of specific IS support through operating the learning cycle from meanings to intentions to purposeful action among the specific group of organizational members.

The Perceived Challenges

It has been the author's experience that SSM is more of a framework of ideas for exploration rather than a step-by-step prescription that practitioners should follow. However, empirical experience shows the practice of this framework does bridge the gap between client and developer in getting the work right when designing interactive systems for ill-structured problem situations involving incomplete and diverse sources of information, and competing demands from numerous stakeholders. If application designers are to get users' work right (Carroll, 2000; Gauss & Weinberg, 1989; Mirel, 2004), they must understand the contextual conditions shaping the intertwined regularities and idiosyncrasies of open-ended inquiries and create software that support the right moves and degrees of agility at the right times and places and for specific purposes. In developing this useful support, HCI specialists and others on a design team need to get their own work right, too; namely, design thinking, and methods of exploration must be in sync with the demands of the complex problem solving.

The integration of SSM into the teaching of HCI represents yet another step of faith which is based on the belief that designing an interactive system (IS) will require attention to the purposeful action which the IS serves, and hence to the meanings which make those particular actions meaningful and relevant to particular groups of actors in a particular situation. In other words, if we wish to create an appropriate IS in the exact sense of the phrase, we must first understand how the people in the situation conceptualize their world. We must find out the meanings they attribute to their perceptions of the world and hence understand which action in the world they regard as sensible purposeful action, and why.

Ideally, it is hoped that our PBL students could enjoy the integration of this activity-centered development process of SSM, into their HCI course conducted through PBL. In reality, our students did experience many difficulties in teamwork throughout the project period. This is largely due to the iterative nature of the process that is quite time-consuming and cognition-demanding for individual team members. They need a lot of emotional support and encouragement to continually invest their time and efforts to carry out the course activities amid the heavy workload of other courses. Indeed, misunderstanding among students and mis-communications between the teacher and the student-groups could easily backfire into students' complaints to call for a halt in the middle of the learning process.

Students do need some breaks to see through the meaning of the course design. As a teacher-designer, experience from the trenches has constantly reminded me of the soft issues among the class to explain the differences between this constructivist approach and the didactic lecture-based mode, and why it is important for them to grow into their future professional careers as software practitioners. However, the practice of PBL, unconfined by discipline boundaries, did encourage my students in the past semesters to accept this integration of SSM to their learning HCI. This is really the biggest encouragement to continue refining the SSM element in the HCI course in the semesters to come.

References

Barrows, H.S. (1986). A Taxonomy of Problem-Based Learning Methods. *Medical Education*, Vol. 20, pp. 481-486.

Carroll, J.M. (2000). *Making Use: Scenario-Based Design of Human-Computer Interactions*. MIT Press.

Checkland, P. & Holwell, S. (1998). *Information, Systems, and Information Systems: Making Sense of the Field*. New York: John Wiley and Sons.

Checkland, P.& Scholes, J. (1999). *Soft Systems Methodology in Action*. New York: John Wiley and Sons

Dreyfuss, H. (1955). *Designing for People*. New York: Simon & Schuster.

Gause, D., & Weinberg, G. (1989). *Exploring Requirements: Quality Before Design*. Dorset House Publishing.

Greening, T. (2000). Emerging Constructivist Forces in Computer Science Education: Shapiong a New Future? In T. Greening (ed.), *Computer Science Education in the 21st Century*, New York: Springer-Verlag, pp. 47-80.

Mirel, B. (2004). *Interaction Design for Complex Problem Solving: Developing Useful and Usable Software*. New York: Morgan Kaufmann.

Ryan, G. (1993). Student Perceptions about Self-directed Learning in a Professional Course Implementing Problem-Based Learning. *Studies in Higher Education*, Vol. 18, pp.53-63.

Vat, K.H. (2001). Teaching HCI with Scenario-Based Design: The Constructivist's Synthesis. Proceedings of the Sixth Annual ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE2001), Canterbury, U.K., Jun. 25-27, pp. 9-12.

Vat, K.H. (2000). Teaching Software Psychology: Expanding the Perspective. Proceedings of the Thirsty-first SIGCSE Technical Symposium on Computer Science Education, Austin, TX, Mar. 8-12, pp. 392-396.