Communication Challenges in Requirements Definition: A Classroom Simulation

Neil C. Ramiller
Erica L. Wagner
School of Business Administration
Portland State University
P.O. Box 751
Portland, OR 97207-0751, USA
neilr@sba.pdx.edu elwagner@pdx.edu

ABSTRACT

Systems analysis and design is a standard course offering within information systems programs and often an important lecture topic in Information Systems core courses. Given the persistent difficulty that organizations experience in implementing systems that meet their requirements, it is important to help students in these courses get a tangible sense of the challenges they will face, whether as Information Systems practitioners or business professionals, in the systems analysis and design process. This article presents a hands-on design game that focuses in particular on the structuring of opportunities for user participation in requirements definition. The game provides a platform for raising pivotal questions about communication, knowledge transfer, and the level and timing of user involvement during systems projects. The exercise has been used and refined over a period of several years in core courses in information technology management at both the undergraduate and graduate levels and in classes in systems analysis and design. The article includes theoretical grounding in user participation issues, background information about the game, specification of the materials needed, step-by-step instructions for conducting the game, and teaching notes to support classroom discussion. These materials are designed to be useful to Information Systems faculty who want to supplement lecture and/or reading material on the subject of systems development.

Keywords: Systems analysis and design, System development life cycle (SDLC), User requirements, User acceptance, Cooperative learning, Simulated environments

1. INTRODUCTION

Systems analysis and design is a standard course offering within information systems programs and often an important lecture topic in Information Systems core courses. Given the persistent difficulty that organizations experience in implementing systems that meet their requirements, it is important to help students in these courses get a tangible sense of the challenges they will face, whether as Information Systems practitioners or business professionals, in the systems analysis and design process. This article presents a hands-on design game that focuses in particular on the structuring of opportunities for user participation in requirements definition. The game provides an opportunity to raise central questions about communication, knowledge transfer, and the level and timing of user involvement during systems projects.

Students are organized into small groups that adopt multiple roles over the course of a simplified "system" development life cycle. Each group begins in the role of *users* with the initial articulation of a business need or opportunity, which they simulate by creating a model using Lego blocks. The Lego models are then put away, and pairs of teams exchange roles as *users* and *analysts* in conversations focused on preparing requirements documents that will give an account of each user team's model. During the subsequent construction phase, *programmer* teams attempt to use these requirements documents to recreate the original models. Acceptance testing follows, during which the entire class evaluates pairs of models – in each case, the original model representing the users' business requirements and the corresponding model created by the programmer team. The final step in the exercise is a post-project review, when the class discusses the challenges that arose during the game, and the instructor draws parallels to problems in system implementation practice.

This exercise has been used and refined over a period of several years in core courses in information technology management at both the undergraduate and graduate levels and in classes in systems analysis and design. Students find the exercise highly engaging, and the divergent mismatches that always surface between "before" and "after" models are the cause of hilarity and good-natured finger-pointing. (See Figure 1a below with a "before" model on the left and the

companion "after" model on the right; the requirements document is in Figure 1b.)

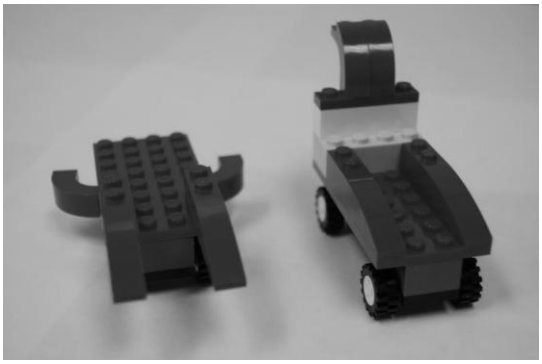

Figure 1a: Before and After Models

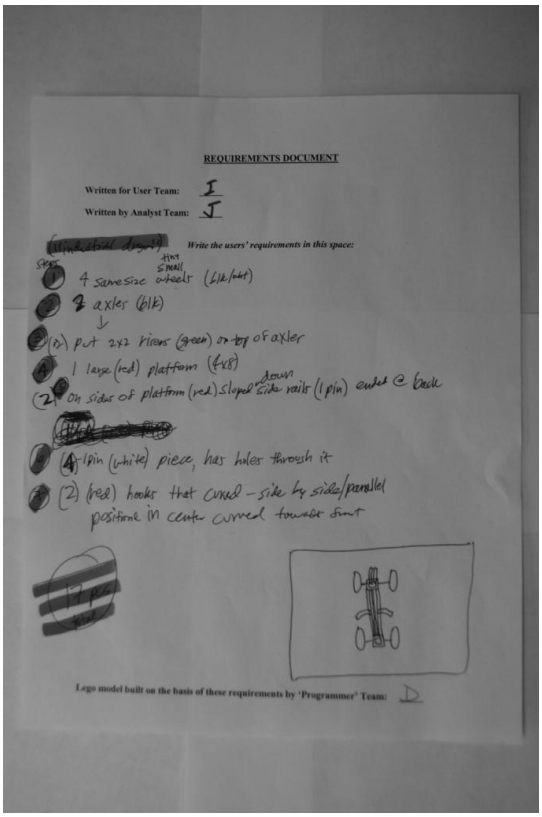

Figure 1b: Companion Requirements Document

The full payoff comes in the final phase, when students, with the instructor's guidance, draw out parallels between the difficulties encountered first-hand in the interpersonal communication of the game and the problems that commonly arise in translating business professionals' requirements via systems analysis for software builders. This also provides an opportunity to explore the implications of alternative project structures for user participation, and to make connections more broadly to issues of IT governance and business-side accountability.

We begin our discussion here with some theoretical grounding in user participation issues, and we then explain how the Design Game helps to surface problems in this domain. After a summary overview of the game, step-by-step instructions are offered for conducting the exercise. Next, we provide detailed teaching notes to help guide instructors in preparing materials, integrating the exercise within a course plan, facilitating the related class discussion, and making the most of the game as a metaphor for real-world challenges in user participation. We conclude with some observations on learning outcomes, based on our experiences in using the game.

2. USER PARTICIPATION IN SOFTWARE DEVELOPMENT

In the 1980s and 1990s system development methodologies relied upon the identification of known requirements (Valusek and Fryback, 1985) in a manner that didn't accurately model the real world as users experienced it (Land, 1982). This often resulted in dissatisfied users who first experienced the information system at installation when it was seen to be too late to make changes (Avison and Fitzgerald, 1995). Research began to reveal how complex the system development process often is, leading to the questioning of some common assumptions. Such assumptions included, notably: that users know precisely what their information needs are and can communicate these easily to system designers (Argyris, 1987); that information needs are static (Land, 1982); and that relationships and communication issues between user and designer are straightforward (Argyris, 1987; Oliver and Langford, 1984). Notwithstanding these early insights, continuing research has documented the fact that companies still struggle with their system implementations, facing user resistance and running significantly over budget and schedule milestones (e.g., Wagner and Newell, 2004; Sauer et al., 2001; Scott and Vessey, 2002).

User participation has been seen as a crucial element for fostering system acceptance. (The Standish Group's annual CHAOS reports have ranked user involvement as the 1st (1994) and 2nd (2000) factor for successful IT project success. See: www.standishgroup.com/.) This is the case not simply because user participation can promote "buy-in," but more importantly because it can help to ensure that the system design ultimately serves the business. Moreover, user participation was not just a response to the "failure of conventional design but it was also based on a belief that users have a right to design their work environment" (Dearnley and Mayhew, 1983: 37). The work of Enid Mumford specifically emphasized the importance of participative system design more generally, and this emphasis has been widely embraced within the context of information system development (Howcroft and Wilson, 2003). Mumford developed the ETHICS methodology (Mumford and Weir, 1979; Mumford, 1995), where system development is seen as inherently complex, requiring negotiations between different stakeholder groups. From this perspective the involvement of multiple groups in negotiations may require more work up-front but is central to system success, so that requirements can be determined and

accommodations made prior to implementation (Mumford, 1983a).

In a different quarter, commentators on evolutionary and agile alternatives in software development began to shed light on the implications of project structure for the actual effectiveness of user engagement in system design (Austin, 2007; Cockburn, 2006; Highsmith, 2002; MacCormack, 2001). One of their central observations is that software development is typically not very much like structural engineering, where the requirements and constraints can be well-understood from the beginning. Accordingly, "structured methods" approaches that assume such idealized engineering conditions and, as a result, sequester user participation in a discrete "requirements determination" stage early in a project, tend to fit the realities of software creation poorly. The more innovative the system in question, the more serious this shortcoming becomes. System development processes in the context of business innovation must instead accommodate discovery and learning, and also openly embrace evolution in requirements. As one of the champions of agile software development remarks, "Agile practices are based on the belief that neither the customers nor the developers have full knowledge in the beginning and that the important consideration is having practices that will allow both to learn and evolve as that knowledge is gained" (Highsmith, 2002: 61).

3. THE DESIGN GAME AS METAPHOR

The Design Game we describe here is motivated by the issues raised in the literature and also by on our own field observations concerning problems of this sort. For example, during the first author's investigations of a systems initiative at a large not-for-profit organization (Ramiller, 2005), the project leader was observed to switch from a highly structured methodology to a more improvisational and agile approach, precisely in order to address problems with limited business-side engagement and users' incapacity for articulating system requirements in an abstract and reasonably complete way. The need for a learning- and discovery-based approach in this case was less a matter of the innovativeness of the system itself, and more a question of the organization's lack of history with major systems projects.

The second author's study of a big-bang ERP implementation (Wagner, Scott and Galliers, 2006) highlighted the challenges of gathering requirements from users who could not envision the depth and breadth of change that would result from the implementation and instead told stories about current work practices and hopes for future efficiencies. Analysts had difficulty translating user stories into technical requirements and then communicating those requirements to the IT professionals. The system that was installed failed to meet the needs of powerful users who felt betrayed by the project team. The analysts were surprised by this reception feeling as though they had done their best with the information that was provided by the users.

Given observations like these, our aim was to create a classroom exercise that could help illustrate the problems that can arise when uncertainty shrouds the business requirements, but where users are nevertheless asked to give a complete and unambiguous account of those requirements up-front. Accordingly, the game presents students with a design challenge and then imposes a set of constraints intended to impede knowledge transfer between students playing the role of "users" and students ultimately responsible for creating a "system" to satisfy those users. Moreover, the structure of the game fosters user uncertainty about requirements and sometimes makes communication and consensus among the users difficult. These are all conditions commonly observed in real systems projects.

More specifically, the game is structured so that the construction of the model meant to satisfy the users' "requirements" actually takes place without the users' presence. Moreover, the device of depriving the users of their own model during the "analysis" phase simulates uncertainty about the requirements by taking advantage of the relative complexity of the models, normal limitations in recall, and differences in what students would remember. Making the users' Lego kits in different assortments complicates the user-analyst interaction, simulating a "language" barrier between the two roles, since users have to describe Lego elements that in some cases are unfamiliar to the analysts.

The exercise gives students the opportunity to engage in a personal way with the communication challenges that arise in the kind of multi-role structures that commonly surround requirements definition and system design. This active approach to learning is, in our experience, more compelling and effective than simply lecturing to students about these challenges. Active learning contrasts with traditional approaches that treat teaching as a matter of information transfer based on abstracted facts, prescriptions, recipes, and formulas (Brown et al. 1989; Bruffee 1993; Christensen et al. 1991; Dewey 1987; Garvin 1991; Whitehead 1929). "We have knowledge, in other words, only as we actively participate in its construction" (Elmore 1991: xii).

As a task-focused exercise, the Design Game contributes to an emerging body of teaching resources addressing differing aspects of the system implementation lifecycle (for example, consider Tyran (2006)), while complementing work that presents more comprehensive life-cycle cases in systems analysis and design (e.g., Bajaj, 2006; Cohen and Thiel, 2010; and Guidry and Totaro, 2011). It also furthers the pedagogical application of student role-playing in the discipline (Mitri and Cole, 2007). The idea to devise an exercise using Legos was drawn from articles written by Schatzberg (2002) and Freeman (2003), who reported on the use of Legos in a systems analysis and design course for a different pedagogical purpose.

In the follow-up discussion, students are invited to consider how the structuring of communication activities in a systems project can help to determine how well or poorly users' needs are met in the organizational acquisition or development of software. This positions the instructor to put user participation in the context of alternative methodologies that textbooks commonly discuss, such as the traditional "waterfall" method, adaptations of the traditional approach like RAD and spiral development, prototyping, and agile strategies. The focus of attention in such a comparative analysis can be on how well each approach can support the

discovery of system capabilities that are both valuable and feasible, through the creation of a feedback process between users and analysts, and "between analysis and design that is used to gain as much information as possible from users" (Dearnley and Mayhew, 1983: 40). More generally, students gain an appreciation for the complexity of systems development and the ever-problematic meaning of "user participation."

If the primary focus of the Design Game is on structure, communication, and knowledge transfer within systems projects, the exercise can also provide a platform for the instructor to launch into larger issues in information-technology management. IT governance is one such topic, which can be entertained to particular advantage in core courses. A useful point of departure is the observation that any given structure for user participation is the result of *choices* that have been made to conduct the project in a certain way. But who made these choices? Senior executives? IT management? Were business-side managers given the opportunity to weigh in?

Organizations that adopt methodologies that limit user participation, especially where the degree of business innovation in a systems initiative is high, may be drawing on inappropriate and out-of-date norms. This presents a related opportunity to discuss how innovation champions must often surmount the barrier of institutionalized (taken-for-granted) thinking (Covaleski & Dirsmith, 1988). The role that organizational politics can play in systems projects also enters in here. Moreover, this can be a good occasion to introduce students to a contrary phenomenon, that being the situation where business-side managers abdicate responsibility for participating in systems projects. This commonly has the follow-on effect that they fail to support their employees' engagement in identifying requirements.

What happens in regard to decision rights and influence roles in systems projects is sometimes symptomatic of governance problems across a wide range of IT management issues (Weill & Ross, 2005). Accordingly, the Design Game can be used as a point of transition for considering this larger topic.

There is another way in which we have used the exercise as a platform for exploring issues that go beyond what the Game itself illustrates directly. This is to follow up with an extended discussion of the nature of user participation, variation in its substance and timing, and how it is changing with the prevalent shift away from custom software development toward the acquisition of packaged software and, increasingly, the sourcing of software as an on-line service.

A good place to start in carrying forward a more in-depth examination of user participation is by acknowledging that it has always been subject to varying levels of intensity (Dearnley and Mayhew, 1983; Avison and Fitzgerald, 1995; Mumford, 1983b) ranging from the *consultative*, where the user is interviewed at some point in the project, to the mid-range *representative* approach involving user spokespeople and analysts in the design process with both groups having a say in the decision making. The most participative approach involves all intended user beneficiaries throughout the design process making decisions based on a *consensus* model (Mumford, 1983b). The appropriate level of participation has always been contingent on circumstances, but students also need to be aware that projects often lapse into a state of 'pseudo-participation' where user involvement is claimed but IT professionals actually make the design decisions (Avison and Fitzgerald, 1995; p 90). (We have observed an amusing echo of this in the Design Game, where programmer teams occasionally announce that they have delivered an "improved" version of the Lego model – mainly because they had too much trouble interpreting the requirements document.)

Enterprise system and other package-based implementations suffer from their own kind of pseudo-participation, where the role of the end-user is commonly limited (Kawalek and Wood-Harper, 2002) and lacking influence (Howcroft and Light, 2006). The perception that the solution has already been chosen and that the design is essentially complete is commonly behind the fact that users are not invited to shape the information system in any significant way. Thus, package implementations often go "full circle back to the early days of customized development when users had little involvement" (Howcroft and Light, 2006: 234) and a "myth of user involvement" (p. 232) lends lip-service to user involvement but actions don't actually support it. The difficulty with this, of course, is that whereas certain matters of design may indeed be settled by the choice of package, the issue of requirements – that is, what the system is supposed to do for the business – remains as current as ever, and still cannot be settled without the engagement of the people who actually know the business. One question students might consider is when such engagement becomes appropriate in the altered lifecycle of package implementation (Markus and Tanis, 2000; Sawyer, 2001).

4. OVERVIEW OF THE GAME

The Design Game is carried out in five steps. A summary follows. Detailed instructions for conducting the exercise appear in the next section.

1. Each team plays the role of a group of *system users*. They identify their "business requirements" by putting together a model using assorted Legos provided to them in a resealable plastic bag.

2. The *requirements definition* phase then pairs off teams, and each team in turn attempts to describe to the other team what their model looks like. This represents the users' effort to define their requirements. The user team does not have access to their Lego model during this phase, which challenges students to remember their model's design and often leads to disagreements among the users about the particulars. The analyst team paired with them prepares a requirements document that attempts to give an account of the user team's requirements. (Figure 2 shows the Requirements Document form that we use.) Step 2 takes place in two parts so that each team in a pair gets to play, alternately, the role of user team and analyst team. By the end of Step 2, a requirements document has been produced and

collected for each of the models created by a user team in Step 1.

3. In the *implementation* phase, each requirements document is given to a team not involved (as users or analysts) in preparing the document during Step 2. That *programmer* team is also given a plastic bag containing an identical assortment of Lego blocks that the user team in question had at its disposal during Step 1. The programmer team then attempts to recreate the original Lego model based on the written requirements. The identical Lego assortment ensures that it is possible in theory – however unlikely it may be in practice – for the programmer team to reproduce exactly the users' original object.

REQUIREMENTS DOCUMENT

Written for User Team: _____

Written by Analyst Team: _____

Write the users' requirements in this space:

Lego model built on the basis of these requirements by 'Programmer' Team: _____

Figure 2: Requirements Document

4. During the *acceptance testing* phase, each programmer team's model is compared to the original model on which it is based, in full class discussion. Deviations are noted by the class, and the user team is invited to accept or reject the resulting design outright, or to suggest a reasonable change order that might correct the problems.

5. During the *post-project review*, in full class discussion students identify the challenges raised by the development methodology.

5. HOW TO CONDUCT THE GAME

The following discussion represents an elaborated version of the lecture notes that we use in running the Design Game. The Teaching Notes in Section 6 provide additional information about preparing the materials, scheduling the game, conducting the game, and leading the follow-up discussion.

5.1 Preliminary Step

1. Assign students to teams. Teams of three or four are generally ideal. Teams of five are generally too large. Because teams will be paired off in Step 2, there must be an even number of teams. Give each team a unique letter designation (A, B, C, etc.).

5.2 Step 1: Users Identify a Business Need (7 minutes)

2. Give each team a set of Lego pieces in a plastic bag, plus a plastic box with the team's letter designation on it.

3. Instruction to students: "Create an object using the following number of Lego pieces. For teams A, C, E (etc.), create an object containing 16 pieces, plus or minus 2 pieces. For teams B, D, F (etc.), create an object containing 22 pieces, plus or minus 2 pieces." (Clarification: A complete wheel, including rim and tire, counts for one piece.)

4. "Give your object a name, reflecting its intended function or purpose."

5. "When you are finished building your model, or I call time, put your Lego object in its box. Put the unused Lego pieces back in the plastic bag, seal the bag, and place that in the box, too. Put the lid back on the box."

   "At no time during this phase should you examine other teams' objects. Also, do not write down anything about your model, draw pictures of any part of it, or take a picture of it."

5.3 Step 2: Requirements Definition

Pair Team A with Team B, Team C with Team D, etc. Paired teams should rearrange themselves so that they are facing one another.

Part 1 (14 minutes)

6. "Teams A, C, E, etc. will continue as *user* teams. Teams B, D, F, etc. will now be *analyst* teams."

7. "User teams: You now have one minute to re-examine your Lego model. Leave your model in the box and do not show it to the team opposite you."

   "Analyst teams: I will now give you a form for use in preparing a requirements document."

8. Call time and collect the boxes from the *users*.

9. "User teams: You must now explain to the analyst team opposite you what your Lego object looks like. You may do this only by speaking (you can also use your hands); you may not write anything down or draw any pictures."

   "Analyst teams: Using the requirements form, prepare a written document that will provide enough information so that a third party will be able to recreate the original object. You may provide written instructions, graphical figures, or both. However, you must not let the users review your requirements document for correctness or, in fact, see it at all."

10. Call time and collect the forms.

Part 2 (14 minutes)

11. At this point, the users from Part 1 become the *analysts*, and the analysts once again become the *users*. Then repeat steps 7 through 10.

5.4 Step 3: Implementation (10 minutes)

12. Assign each team a requirements document and the unused bag of Legos that matches the kit originally used by the pertinent user team. Given the pairings in Step 2, possible assignments for different total numbers of teams include these:

    For a 6-team configuration:
    A to E, E to A
    C to F, F to C
    B to D, D to B

    For an 8-team configuration:
    A to E, E to A
    C to G, G to C
    B to F, F to B
    D to H, H to D

    For a 10-team configuration:
    A to G, G to A
    B to F, F to B
    C to H, H to C
    D to I, I to D
    E to J, J to E

13. "Each team will now play the role of programmers. Based on the requirements document, you will attempt to create an object that matches the original Lego model for which the requirements were written. Do not seek assistance from either the *user* team or the *analyst* team who were involved in creating those requirements."

    "When you finish or time is called, turn in your Lego object to me, along with the requirements document and the unused Lego pieces. (Please seal the unused pieces in the plastic bag.)"

14. Collect the models and materials.

5.5 Step 4: Acceptance Testing (Full Class Discussion)

15. Compare each programmer team's object to the original users' model and lead an evaluation and discussion of how closely the two objects relate. Invite the user team to "accept" or "reject" the model that was built for them, based on how closely it satisfies their requirements.

5.6 Step 5: Post-project Review (Full Class Discussion)

16. Engage the entire class in a discussion about the challenges they faced in performing the user, analyst, and programmer roles. Draw parallels between difficulties that students identify in the Game and problems that commonly occur in connection with user participation (and non-participation) in systems development projects. Suggestions for such a discussion are included in the teaching notes for this case. Themes that typically surface include the difficulties of developing a shared language across roles; challenges in creating an effective mode of representation; problems in reaching user consensus; the lack of interaction between users and builders; and alternative project structures that could make for more effective communication.

6. TEACHING NOTES

6.1 Materials

Preparing the materials needed for the game is a relatively straightforward matter. We first acquired a large supply of Lego pieces, in considerable variety, and then created discrete Lego kits in identical pairs. These same kits have continued to serve over several years and many uses. Every kit contains approximately 35 pieces, several more than is required in the students' model. As remarked, the kits differ across pairs, in order to add further challenge to the user-analyst conversation. Each kit is contained in a re-sealable plastic bag. At the beginning of a game, one kit belonging to each identical pair is placed into an opaque box, and the matching kit is set aside for the programmers' use in Step 4. (As the instructions note, the box is used to hide away the users' model, once it is completed.) Finally, we prepare in advance copies of the simple User Requirements Document form show in Figure 2.

6.2 When to Schedule the Game

The Design Game has been successfully deployed as a start-of-term ice-breaker in core information-systems courses. Although this certainly has value in getting a class off to an engaging start, we have concluded that where students lack personal experience with the complexity and difficulty of systems initiatives, they will at this point in the term also lack the context needed for understanding the issues which the exercise illustrates. Accordingly, we now generally conduct the exercise relatively late in the term, in both core courses and systems-development courses, after students have had some exposure to design and implementation issues and the concept of the system lifecycle.

In core courses we have also positioned the Design Game as a bridge between the topics of user participation and information-technology governance. As noted, in

discussing the results of the exercise we raise the point that meaningful user participation is a function of both project structure and management support. Hence, users can be "structured out" of a project; alternatively, they can get left out when their own managers' abdicate business-side responsibility. Other matters of organizational concern in the management of information technology, such as IT project prioritization and selection, are also subject to the same kinds of dysfunctional behavior.

6.3 Students' Advanced Preparation

There is no up-front preparation for the students to complete before the simulation. It can be helpful to have students read ahead of time about alternative systems-development methodologies. On the other hand, we have found the Game to be a compelling introduction to the topic of user and business-side involvement in systems initiatives, with relevant reading then to follow. Homework can also be assigned after the fact, and may be especially appropriate if classroom time for discussion during Step 5 is limited. (It can be based on some variation of the discussion questions we note below.)

6.4 Group Size

As noted, the exercise is based on small groups that shift between user, analyst, and programmer roles during the course of the game. Groups of three are probably ideal, although groups of four can also work well. Pairs of students will typically not produce sufficient within-team variety and complexity in the communication, and teams of five or larger inevitably leave certain students sitting on the sidelines.

6.5 Duration of the Game

The exercise is designed to be completed in a single class session of at least 90 minutes, although an additional 20 minutes will sustain a richer and more extensive discussion in Step 5. A break after Step 2 of some 10 minutes is a good idea, not only to give the students a chance to refresh, but also to allow the instructor to set up the materials (matching Lego kit bags and requirements documents) for the "programming" phase of the Game. The exercise has also been conducted over the course of two shorter class sessions of 50 minutes each. This requires the instructor to keep the original "user" models intact, in their boxes, for comparison with the later models created in the second class. Alternatively, digital photos of the "before" and "after" models can be taken at the appropriate time and then displayed via projector at the next class period.

6.6 Lessons Learned in Running the Game

The exercise is logistically rather involved, so the instructor must be sure to have the students' undivided attention prior to discussing each phase. When students are given the Legos, they tend to get excited and don't always follow what they are supposed to be doing. The strictures to the user teams in Step 1 about not creating documentation for their own models and not examining other teams' models during this phase require particular emphasis, if the game is to produce interesting mismatches in the end. It is also helpful to emphasize that the written requirements form is the only source of information during the "programming" step. The time limits we recommend for each step do not only serve to impose schedule pressure – a realistic factor seen in actual systems projects – but also minimize students' ability to get into the kind of mischief that can undermine the game's effectiveness. On the other hand, it is important to allow sufficient time for students to compare the before and after versions of the models. It is possible to get the class to rank pairs of models in terms of the satisfaction of user requirements. As there is plenty of "blame" to go around in the less successful cases – an important practical observation in its own right – there is generally little possibility for feelings to be hurt, although sensitivity in this regard is in order. Finally, it is a good idea to set aside enough time for an expansive discussion in Step 5 (see the following section). As we have noted, where this is not possible follow-up homework can be assigned.

6.7 Leading the "Post-Project Review" (Step 5)

We normally structure the closing, full-class discussion (Step 5, the "Post-project Review") with a short sequence of questions that begins by getting the students to reflect personally on the challenges they encountered during the exercise. Along the way the instructor will draw parallels between the contrived barriers introduced in the Game and real barriers that participants encounter in actual system projects. The discussion culminates in a consideration of the Game's implications for alternative structures for user participation in systems initiatives.

We introduce the Post-project Review by remarking that this is something managers set out to do on practically every software project, with the best of intentions, but then often never do in the end. A post-project review takes considerable time and energy, and when projects run over schedule and budget (which they still commonly do), managers are reluctant to invest in it. Moreover, when project outcomes are problematic (which they still often are), participants can be anxious to get on to the next thing, or perhaps to clear out altogether, before the inevitable fallout. "Nevertheless," we announce, "we will undertake a post-project review in the present case," because it is a vital organizational learning opportunity. It's a chance to reflect on the process everybody went through, to decide what was good and bad about it, and to figure out how things might be done differently the next time.

Questions 1a and 1b: What difficulties arose for the analyst teams in attempting to prepare the written requirements document based on the users' verbal description of what they wanted? What frustrations did the users experience in trying to communicate with the analysts? In exploring these questions, students often point to difficulties in coming up with a common language for describing the Lego pieces. This trouble can arise within the teams as well as between users and analysts. The instructor can note how the interaction at this point in the game simulates the project situation where users and analysts can't engage around a common object (like a prototype) that represents what the users want. Instead, the parties are trying to move from the users' vision toward some representation that takes an entirely different form. In software development, that representation is often a

graphical or textual abstraction like a process model that the users will not understand. Conversely, analysts can have trouble understanding the business-domain language of the users.

Denying the users the option of writing and/or drawing and denying the analysts the opportunity to review their written document with the users are both contrivances, but they are not done simply to make the task difficult. Both of these conditions help to simulate the fact that users in software projects are typically not in charge of written specifications and, moreover, almost never understand the design formalisms that analysts use.

Students also sometimes note problems with the user team remembering what the model looked like and agreeing on its details. Although this result is produced artificially within the exercise by denying the users access to their model, it reflects the very real difficulties that user representatives sometimes have both in reaching consensus and in developing a completely clear vision of their requirements up-front.

We also sometimes observe, and remark on, variations in user team behavior during the analysis step. Specifically, we have noted three styles, broadly speaking, of user representation. In *collaborative* teams the students largely share a common vision and all students participate in articulating it in a well-orchestrated fashion for the analyst team. In *collective* teams, all students participate in the user-analyst conversation, but they tend to disagree with one another about details of their model. Commonly, this situation leads to fragmented conversations between individual users and individual analysts and, ultimately, a disjointed requirements document. In *lead-user* teams one student dominates the interaction on the user side, with the other user students deferring to that student's "expertise" or, perhaps more likely, dominant personality. The "after" model in such a case is not typically a superior match to the original. We make the point, then, that when lead users dominate requirements specification in real projects, the resulting system doesn't necessarily fit the business better, since lead users may be unrepresentative of, or less knowledgeable than, other users.

It is also fruitful to ask students whether the second user-analyst conversation (in Part 2 of Step 2) was easier. Most students agree with this. The instructor can then point out that the models that are the subject of the second conversations are on average more complex, since they are larger. (See the specifications for model sizes described in the main article.) The correlation between size and complexity is not perfect, of course, but students intuitively grasp that the two will be associated. What accounts, then, for the second part of Step 2 tending to be easier? The instructor has an opportunity, here, to point to *process learning* between the two parts of Step 2, an effect that is notable as real projects progress, provided that there is not a lot of turnover in personnel.

We have sometimes asked students if having more time for the user-analyst conversation would have made a difference. (We have also asked this question in connection with the programmers' task. See below.) Time pressures, of course, are an ever-present factor in real projects. Students' responses to this question are mixed. Some students will insist that they could have used more time. Other students will argue that extra time would have made little or no difference. Problems in user recall or finding a common language to use with analysts can make extra time moot. We have likened this to trying to have a conversation on a cell phone with a really bad connection: No amount of additional time on the line will make the conversation any more sensible. Just about everyone can relate to this, because just about everyone has hung up on a call under these conditions.

Questions 2a and 2b: What difficulties arose for the programmer teams in trying to create an object based on the written requirements document? What factors may have played a role in determining how close the programmers got in reproducing the users' original object? Students' reactions to these questions typically focus on problems in the documents themselves. Lack of clarity about the identities of pieces and their interrelationships (the language problem, again), incompleteness in the specification, and contradictions are all commonly noted. When the instructor asks whether students think pictures or words work better to communicate the users' requirements, the most common response is that both together seem helpful, but only to the extent that each is executed skillfully. Where the programmers' model is quite different from the users' original model, the user and analyst teams involved readily revisit the issues associated with Question 1 (see above), and the good-natured finger-pointing that ensues can give the instructor an opportunity to discuss the distributed nature of accountability in such situations. It also provides an opening to observe that the structuring of the work can be as much to blame as any of the actors.

It is during consideration of the programmers' challenge that students also most commonly begin to reflect on the comparative design of the different models. User models that have relatively clean and symmetrical forms uncomplicated by ornamentation are usually reproduced by the programmers with higher fidelity. The instructor can note that simplicity is not *per se* a virtue in itself, but where complexity may in fact be appropriate in a design; it increases the challenge of knowledge transfer.

To further elevate the critique above the level where students nit-pick the documents, the instructor can call attention to the central fact that all the programmers have to consider is the document. Even in circumstances where a standardized methodology prescribes a consistent form for such documents – which is far from the case in the Game where the students, acting in their role as analysts, must improvise the documentation approach – they offer a narrow vehicle for the representation of requirements. This is especially true where users are experiencing significant uncertainty to begin with, or where there are difficulties in users and analysts communicating.

Noting how the requirements, in such problematic form, had been "thrown over the wall" to the programmers provides the segue to the next discussion question. Instead of putting the programmers utterly at the mercy of a document, how might their work have been better supported?

Question 3: How might things have been done differently, so as to make the task easier and/or more successful? We ask the students to assume that the initial conditions remain

the same, specifically, that users do not get to document their own models or to look again at their creations after Step 1. Students will sometimes then propose that things would have gone better if the instructor had provided them with a standardized format for organizing the requirements document and perhaps a visual listing of the possible Lego components. Such a proposal constitutes, more or less, a "structured methods" approach to improving the process. It is good to observe at this point that the result is likely to be a more consistently readable requirements document, but that this will not help much with uncertainty the users may have about the requirements themselves. The class discussion will then move rather quickly to a proposal to merge the roles of analyst and programmer, and to blend the work of analysis and programming so that the user team can converse with the analyst/programmer team as the latter attempt to recreate the users' original model. The model, as it emerges, would become the medium for this undertaking, and the requirements document would be dispensed with. This corresponds to a prototyping or agile approach to development, and moves the process from discrete stages to an evolutionary trajectory.

Question 4: While the Design Game is most directly a metaphor for software development, does it hold any larger implications for IT management? This question is less a lead-in to student discussion and more a way to frame some general instructor remarks about responsibilities and accountability in the IT domain. This is a good way to wrap up the Post-Project Review. In regard to project methodologies that structure-out effective user participation, we have found it both amusing and helpful to present students a version of the famous tree swing cartoon. (Googling "tree swing cartoon" will produce several versions of this.) This cartoon shows a succession of increasingly impractical and ridiculous designs, as the tree-swing project gets handed off from project sponsor, to analyst, to programmer, and the like. The punch line shows that the user wanted a tire swing, which doesn't remotely resemble what everyone else was working on.

We also point out, however, that although sometimes the project structure will accommodate effective user participation, the business side may abdicate responsibility. Hence, effective user participation is a two-way street. To support this point, a specific Dilbert cartoon provides an entertaining summation. It offers the following dialog between analyst and user (Adams, 2006: 86):

> *Analyst: I'll need to know your requirements before I start to design the software. First of all, what are you trying to accomplish?*
>
> *User: I'm trying to make you design my software.*
>
> *Analyst: I mean what are you trying to accomplish with the software?*
>
> *User: I won't know what I can accomplish until you tell me what the software can do.*
>
> *Analyst: Try to get this concept through your thick skull: The software can do whatever I design it to do!*
>
> *[pause…]*
>
> *User: Can you design it to tell you my requirements?*

In a course that significantly explores the topic of information-technology governance, as many core classes do, this pairing of the tree-swing and Dilbert cartoons provides a nice segue' into broader questions of IT management responsibility that reach beyond the domain of system implementation.

7. CONCLUSIONS

The Design Game enhances information systems education by giving students the opportunity to engage, in a personal way, in a task central to the application of information technology: the communication of design requirements. Through rotating role assignments the exercise also helps students to see this task from diverse perspectives, and to appreciate the challenges that arise in connection with the different jobs that people do in systems development. A representational student quote shows evidence of learning:

> *"One take-away that I learned from this assignment would be realizing how a vision of an object can be translated and skewed as it gets passed along through the analysis process from user to analyst to programmer."*

The abstract discussions of methodologies and user involvement that typically appear in systems textbooks tend to fall short, when it comes to convincing students that good design indeed depends on effective management and personal commitment to the often hard work of communication. For example:

> *"The biggest thing I will take away from the [game] is how difficult it can be to communicate with a client. I believe that both sides wanted to have a perfect transfer of information but in the end we fell short. It was a little shocking to see how difficult it is to explain how to build something so small that is comprised of so few pieces…Keeping this in mind I will make sure to take the time to formulate thoughtful questions and do my best to involve the client in order to better ensure that I receive the best possible information".*

And another student reflects:

> *"This assignment has merit - it is very close to real life situations that analysts deal with on a daily basis*

The Design Game makes these crucial insights tangible in a way that is both entertaining and compelling.

8. ACKNOWLEDGEMENTS

The authors would like to thank their students over the years for their enthusiastic engagement with the design game.

9. REFERENCES

Adams, S. (2006), Try Rebooting Yourself: A Dilbert Collection. Riverside, NJ: Andrews McMeel.

Argyris, C. (1987), "Some inner contradictions in Management Information Systems", in H Lucas, et al., (eds.), The Information Systems Environment, Amsterdam: North Holland; reprinted in R Galliers (ed), Information Analysis: Selected Readings, Wokingham: Addison-Wesley, op cit. pp. 99-111.

Austin, R.D. (2007), "CMM versus Agile: Methodology wars in software development," Boston, MA: Harvard Business School, Case #9-607-084.

Avison, D., and Fitzgerald, G. (1995), Information Systems Development: Methodologies, Techniques and Tools, 2nd edition, London: McGraw-Hill.

Bajaj, A. (2006), "Large scale requirements modeling: An industry analysis, a model and a teaching case," Journal of Information Systems Education, 17(3), pp. 327-339.

Brown, J.S., Collins, A., and Duguid, P. (1989), "Situated cognition and the culture of learning," Educational Researcher, 18, pp. 32-42.

Bruffee, K.A. (1993), Collaborative Learning: Higher Education, Interdependence, and the Authority of Knowledge, Baltimore: Johns Hopkins University Press.

Christensen, C.R., Garvin, D.A., and Sweet, A. (eds.) (1991), Education for Judgment, Boston, MA: Harvard Business School Press.

Cockburn, A. (2006), Agile Software Development: The Cooperative Game, 2nd edition, Reading, MA: Addison-Wesley Professional.

Cohen, J.F., and Thiel, F.H. (2010), "The Rescue911 Emergency Response System (ERIS): A systems development project case," Journal of Information Systems Education, 21(2), pp. 149-157.

Covaleski, M.A., and Dirsmith, M.W. (1988), "An institutional perspective on the rise, social transformation, and fall of a university budget category." Administrative Science Quarterly, 33(4), pp. 562-587.

Dearnley, P.A., and Mayhew, P. J. (1983) "In favor of system prototypes and their integration into the system development life cycle," Computer Journal, 26(1).

Dewey, J. (1987), Experience and Education, reprint edition, Scribners; original publication: New York: Macmillan, New York, 1938.

Elmore, R.F. (1991), "Foreword," in C.R. Christensen, D.A. Garvin, and A. Sweet (eds.), Education for Judgment, Boston, MA: Harvard Business School Press, pp. ix-xix.

Freeman, L.A. (2003) "Simulation and Role Playing with LEGO Blocks," Journal of Information Systems Education 14(2), pp. 137-144.

Garvin, D.A. (1991), "Barriers and gateways to learning," in C.R. Christensen, C.R., D.A. Garvin,, and A. Sweet (eds.), Education for Judgment, Boston, MA: Harvard Business School Press, pp. 3-13.

Guidry, B.N., and Totaro, M.W. (2011), "Convention Center Management: A systems analysis & design course project," Journal of Information Systems Education, 22(1), pp. 15-17.

Highsmith, J. (2002), Agile Software Development Ecosystems. Reading, MA: Addison-Wesley Professional.

Howcroft D., and Light, B. (2006), "Reflections on issues of power in packaged software selection," Information Systems Journal, 16(3), pp. 215-236.

Howcroft, D., and Wilson, M. (2003), "Participation: 'Bounded freedom' or hidden constraints on user involvement," New Technology, Work and Employment, 18(1), pp 2-19.

Kawalek P., and Wood-Harper, A.T. (2002), "The finding Of thorns: User participation in enterprise system implementation." ACM SIGMIS DataBase, 33(1), pp. 13-22.

Land, F. (1982), "Adapting to changing user requirements," Information & Management, 5(2), 1982, pp. 59-75. Reprinted in Galliers (ed.) 1987 op cit.: pp. 203-29.

MacCormack, A. (2001), "How Internet companies build software," MIT Sloan Management Review, Winter 2001, pp. 75-84.

Markus, ML & Tanis, C (2000), "The Enterprise Systems Experience - From Adoption to Success", in Framing the Domains of IT Management: Projecting the Future through the Past, ed. RW Zmud, Pinnaflex Publishing, Cincinnati, OH, pp. 173-207.

Mitri, M., and Cole, C. (2007), "A systems analysis role play case: We Sell Stuff, Inc.," Journal of Information Systems Education, 18(2), pp. 163-168.

Mumford. E., and Weir, D. (1979), Computer Systems in Work Design – the ETHICS method. New York: Wiley.

Mumford, E. (1983a), Designing Human Systems. Manchester, UK: Manchester Business School.

Mumford, E. (1983b), Designing Participatively. Manchester, UK: Manchester Business School.

Mumford, E. (1995), Effective Systems Design And Requirements Analysis. Basingstoke: MacMillan.

Oliver, I., and Langford, H. (1984) "Myths of demons and users: Evidence and analysis of negative perceptions of users," Proceedings: Australian Computer Conference, Sydney, NSW, November 4-9, Australian Computer Society, 1984, pp. 453-463; reprinted in Galliers (ed.) 1987 op cit.: 113-23.

Ramiller, N.C. (2005), "Applying the sociology of translation to a system implementation in a lagging enterprise," Journal of Information Technology Theory & Applications, 7(1), pp. 51-76.

Sauer, C., Liu, L. and Johnston, K. (2001), "Where project managers are kings," Project Management Journal, 32(4), pp. 39-49.

Sawyer, S. (2001), "A market-based perspective on information systems development," Communications of the ACM, 44(11), pp. 97-102

Schatzberg, L., (2002), "Applying Bloom's and Kolb's theories to teaching systems analysis & design," Proceedings of ISECON, the Information Systems Educator Conference, 2002.

Scott, J. and Vessey, I. (2002), "Managing risks in enterprise systems implementations," Communications of the ACM, 45(4), pp. 74-81.

Tyran, C.K. (2006), "A software inspection exercise for the systems analysis and design course," Journal of Information Systems Education, 17(3), pp. 341-351.

Valusek, J. R., and Fryback, D. G. (1985), "Information requirements determination: Obstacles within, among, and

support Natural between participants", Proceedings: End-User Computing Conference, Minnesota, ACM Inc., 1985, Reprinted in Galliers (ed.) 1987 op cit.: pp. 139-51.

Wagner, E., Scott, S., and Galliers, R. D. (2006), "The creation of 'best practice' software: Myth, reality and ethics," Information and Organization, 16(3), pp. 251-275.

Wagner, E. and Newell, S. (2004), "'Best' for whom?: The tension between best practice ERP packages and the epistemic cultures of an Ivy League university," Journal of Strategic Information Systems, 13(4), pp. 305-328.

Weill, P., and Ross, J. (2005), "A matrixed approach to designing IT governance," MIT Sloan Management Review, 46(2), pp. 26-34.

Whitehead, A.N. (1929), The Aims of Education and Other Essays, New York: Free Press.

AUTHOR BIOGRAPHIES

Neil Ramiller is a professor in the Management area at Portland State University's School of Business Administration. His primary research activities address the management of information-technology innovations, with a particular focus on the role that discourse plays in shaping innovation processes within organizations and across inter-organizational communities. He is also interested in the social construction of information technology scholarship. He has presented his work at a variety of national and international conferences, and his articles have appeared in a number of journals, including *Journal of the Association for Information Systems*, *MIS Quarterly*, *Information & Organization*, *Information Technology & People*, *Organization Science*, *Journal of Management Information Systems*, *Communications of the AIS*, and *Information Systems Research*. He is a member of the editorial boards of *JAIS*, *Information, Technology & People*, and *Information & Organization*, and a past associate editor for *MIS Quarterly*. Dr. Ramiller's Ph.D. is from the Anderson School at UCLA.

Erica Wagner is an associate professor in the Management area at Portland State University's School of Business Administration. She earned her Ph.D. from the London School of Economics and has an undergraduate degree in accounting. She has previously taught at CornellUniversity and The London School of Economics. Her research interests focus on the ways software is 'made to work' within different organizational contexts, with particular emphasis on how work practices are designed into artifacts, standard processes, and methods of accounting. Her research has been published in a variety of outlets including The Journal of the Association for Information Systems, Information and Organization, Communications of the ACM, and the Journal of Strategic Information Systems. Dr. Wagner's paper entitled "The creation of 'best practice' software: Myth, reality and ethics", was awarded "Best Research Paper 2006" by leading scholars in her field. In addition, she was one of four faculty members across Cornell University to receive a 3-year grant from the National Science Foundation's (NSF) Digital Government project (2005) toLanguage Processing Support for eRulemaking.

Information Systems & Computing
Academic Professionals

STATEMENT OF PEER REVIEW INTEGRITY

All papers published in the Journal of Information Systems Education have undergone rigorous peer review. This includes an initial editor screening and double-blind refereeing by three or more expert referees.

ISSN 1055-3096