7. On using materiality in information systems development: a research brief

ANDREA CARUGATI
UNIVERSITY OF AARHUS

Abstract

This research brief presents a discussion on the use of the concept of materiality and material knowing in information systems development (ISD). The discussion addresses some of the practical problems still plaguing ISD, augmenting existing ISD methodologies with contributions from systems theory and in particular the idea of inquiring systems. The discussion builds on different contemporary concepts that are rooted in the inquiring systems idea: the notion of stakeholders (designer, client, user and their interchanging roles), the notions of a boundary object and boundary spanners and the notion of materiality as a scaffold of knowledge. Through the example taken from a case study of a complex and innovative systems development, we outline two design principles to be embedded in modular fashion in ISD processes: 1) whenever possible, start ISD efforts by developing a graphical simulator of the material environment of the users; and 2) embed and use the simulator as a proper boundary object.

Introduction

The development of proprietary information systems has been and continues to be a very complex activity marked by few successes and clamorous failures (Beck 1999; Remenyi et al. 1997). Despite a continuous search for better techniques, as Avison et al. (1995) point out, the problem is not in the tools used but in the lack of attention to organisational and individual issues and their interaction with technology. Responding to this situation, research and practice have begun to consider the information system development (ISD) process as a social rather than a technical problem (Gibson and Singer 1982; Winter et al. 1995). A social constructivist approach to ISD, in which systems and requirements emerge
from the interaction of multiple stakeholders, has begun to be preferred to an objectivist approach that considers requirements as exogenous factors, existing outside the interaction of individuals. Hence, the attention has shifted on to user participation, which has risen to become one of the most important components of system success (Avison et al. 1995; Barki and Hartwick 1989, 1994; McKeen and Guimaraes 1997; Winter et al 1995). This interaction between users and other different groups (for example, developers, managers and analysts) can be considered as the minimal activity to bring about just enough knowledge exchange to make the system requirements emerge. Within this view, it has become very important to address the issue of how to facilitate the process of knowledge creation and exchange among the stakeholders involved in ISD projects. Starting from these premises, knowledge issues in ISD have also been addressed and treated both explicitly (for example, Beck 1999) and implicitly (Avison and Fitzgerald 1995). This is because the successful development of information systems depends on the assimilation and combination of knowledge coming from different domains and on the intangible, cognitive and social nature of some ISD goals.

Yet, despite the advances in research and practice in ISD, the situation does not seem to improve. Informal discussions with programmers in leading Danish software houses (most of them using SCRUM) have pointed out that they still have problems engaging the customers in the discussions and therefore can use agile methods in only a sub-optimal way:

Sure there are tons of methods out there and why do you think that is? Because they have not found the answer yet. Take SCRUM, for example, the one we use. The basic idea is good but no customer wants to waste time in talking with us and they expect you to be the professional with the answers. (Programmer, software house, Denmark, 2008)

So, while it is widely accepted nowadays that the pursuit of knowledge exchange in ISD is the way to go, apparently the way it is done in practice is too cumbersome to be practical. Knowing about the inquiring cycle (Carugati 2008), about the boundaries (Levina 2005; Levina and Vaast 2005) and about the iterations (Lindstrom and Jeffries 2004) provides invaluable elements but something more needs to be done. This something has to decrease the involvement of the customer while favouring the knowledge exchange.

It is the belief of the author that this missing element can be found only in what is under the eyes of the programmers: the material environment of the customer. The missing element in the ISD discourse is the role of the environment of the users in the development process. Most research focuses on office applications (Kellogg et al. 2002; Levina 2005; Levina and Vaast 2005) and these settings represent only a small part of the field of uses of information systems. In
industrial settings (for example, manufacturing versus services), the physical layout where the information system has to be used plays a key role in how the system has to be and how the users can capitalise on their knowledge to interact with the developers. According to Orlikowski (2006), human action depends on physical artefacts such as buildings, machines, vehicles, and so on. In other words, what we do depends on the material world that surrounds us. What we do and how we do it are related to what we know (Churchman 1971) and also to how we learn (Cohen and Levinthal 1990). These three events are not, however, sequential but rather happen in a synchronous fashion. Orlikowski defines this situation as ‘material knowing’, indicating that knowledge is intimately connected with the materiality that surrounds us. While the work on boundary objects (Carlile 2002) focuses the attention on (material) objects created on the boundaries to facilitate the passage of knowledge, the way in which we read Orlikowski (2006) here refers mostly to the materiality of the environment existing around the users.

A way to improve the ISD process is therefore to take into account not only the creation process and the object created but also the materiality of the situation that, to quote Orlikowski, scaffolds the problem definition. The goal of this research brief is therefore to present a way to introduce the notion of materiality and boundary objects to ISD.

The chapter briefly introduces the notions of materiality and boundary objects and, using a case to anchor the discussion, presents the concept of material software objects. Throughout the chapter, we refer to experiences gained from the development of an inventory-control system for a shipyard (SteelCo). The SteelCo case will be described in the next section to provide the basic elements for the discussion.

**The SteelCo case**

SteelCo, a multinational shipyard located in Denmark, is known worldwide for producing very sophisticated vessels with designs at the forefront of the market. During the period 1999–2002, it earned about €1 billion per annum, employing about 3000 people. SteelCo builds vessels that are among the largest container carriers in the world. Building a container ship of the maximum size entails the cutting and welding of more than 150,000 steel elements, 30,000 larger components and 11,000 pipes. Consequently, this production process places heavy demands on planning and logistics. One of the key departments in SteelCo is the steel-plates inventory since the near totality of steel elements
comprises parts cut out of these large plates. The inventory is the centre of reception and distribution of steel plates and therefore a key element in the internal logistics of SteelCo.

Figure 7.1a shows the layout of the inventory (1). Starting from the bottom, we have the sea with the ship from the supplier (11) that delivers the steel plates. The plates (Figure 7.1b) are stacked in 256 piles organised like a chessboard (8–32). Each pile could have up to 200 plates. Each plate is unique and identified by a product number written on it. There are two yellow cranes that move the plates (Figure 7.1b). The cranes move on the same rail and therefore cannot pass each other. On the bottom left (Figure 7.1b) is shown the beginning of the preparation line where the plates are cleaned and painted and prepared for the plasma cutters.
The crane operator does not know exactly in which pile a specific plate is located; he or she only knows that it can be found in one of four to six different piles. It is left to the experience and memory of the crane operators to know and find where a specific plate is. Given the production needs of the plasma cutters, the job of the crane operators is to find the right plates in the piles by moving the ones above to other piles. The main problem of the inventory is therefore the location of the plates and their movements so as to ensure a continuous flow of work for the plasma cutters of the right plates at the right time and to facilitate future work in the inventory.

The inventory is hierarchically organised. Those responsible for the inventory tour the plasma cutters in the morning and prepare the delivery lists based on the needs of the plasma cutters and the actual production plan. This list (a sheet of paper) is given to the crane operators, who move the plates around in the inventory, dig up the delivery plates and place them on the preparation line. The plates’ delivery sequence and the movements of the other plates are completely up to the crane operator.

SteelCo has, over the years, experimented with information technology (IT) to find and exploit many possible ways to augment the efficiency and effectiveness of the inventory management. The characteristics of the setting have, however,
defied all attempts. Among these characteristics, the most commonly mentioned were the weather, the bending of the plates, the errors in reading the plates’ identification numbers, cranes picking up multiple plates in one lift, plates becoming wedged in the conveyor belt and even cranes’ wheels becoming ‘square’ over time. Among the many solutions (automated plate reception, plate position monitoring, a crane-positioning system, and so on), the one that most concerned the management of SteelCo was the implementation of a software tool that could guide the crane operators in deciding the sequence and destination of the plates’ movements. The goal was to minimise the number of movements, minimise the hours worked, minimise work in relation to changes in the production plan and minimise reaction time in response to orders while at the same time respecting the physical constraints of the layout. For this problem, there did not exist (at least to the knowledge of the actors) ready software solutions acquirable on the market or solutions to be customised.

For this reason, it was decided to try to solve the problem with tailor-made software. The idea was to include in the software an algorithm based on the combinatorial optimisation technique. This technique promised to deliver a solution that could fit multiple and changing goals. Combinatorial optimisation is a very complex technique for which there are no known development methods for industrial use. The main difficulty with its use is that—as far as the actors are concerned—the algorithm has to be developed from scratch to completion before the system can be tested. Furthermore, since the work on the inventory was completely manual, the problem of using software to guide the crane operators was a completely new problem for the different actors. Thus, the situation was that there was an unsolved problem, the approach requested of the developers was innovative and the development activities were unstructured and not predictable in advance. At the outset, it was considered that the inventory project would take one year to complete.

**Materiality, boundary objects and ISD**

In her 2006 article on material knowing, Wanda Orlikowski makes an argument for taking materiality seriously in IS research. To stress the importance of the argument in the article she quotes Latour: ‘There exists no relation whatsoever between the material and the social world, because it is the division that is first of all a complete artifact. To abandon this division is to rethink the whole assemblage from top to bottom and from beginning to end’ (Latour 2004:227).

In Orlikowski’s view, the material characteristics of the world surrounding us are an integral part of what we do and what we know. In this view, material forms, artefacts, spaces and infrastructure play critical roles in everyday practices.
and the knowledge embedded in the practices. As Latour points out, however, researching in the integration of materiality and practices in fact means going back to the basis of human sciences.

The view of Orlikowski (2006) is in line with that of this chapter, because she states that her arguments fit a performative view of knowledge—a knowledge that is not static but a dynamic and continuing social accomplishment. This view fits perfectly in the ISD landscape, where developers and users engage continuously in activities that have knowledge creation as output (Carugati 2008). What Orlikowski so well articulates and adds to the discussion about knowledge is that knowing is not only emergent, embodied and embedded (three concepts that are now slowly being integrated in ISD efforts), it is also material.

By knowledge being material, Orlikowski means that everyday practices and therefore the knowing attached are ‘deeply bound up in the material forms, artifacts, spaces, and infrastructures through which humans act’ (p. 460).

In the SteelCo case, for example, the actions of the crane operators are highly dependent on readily apparent objects such as cranes, plates and ships as well as by less conspicuous ones such as the production plan or the information infrastructure of the shipyard. As Orlikowski points out, at the ‘level of conceptualising and theorising, we tend to disregard this knowing, and render our accounts of knowledge in organisations without attention to material matters’ (p. 460). This influences all aspects of IS research and therefore also ISD. This is especially true since the majority of recent cases on ISD that take knowledge issues into account are grounded in ‘office’ cases (for example, Levina 2005) where the materiality issue is much less visible (but by no means irrelevant).

Forgetting materiality seems to be quite widespread nowadays since the dominating paradigm in ISD is human centric, focusing largely on human interpretation of actions while technology tends to take a backstage role (Orlikowski and Iacono 2001). This problem leads us back to Latour’s statement about the lack of attention to materiality as an inseparable part of human action.

As one of the main problems in ISD is the exchange of knowledge across different parties—typically users and developers (Carugati 2008; Levina 2005)—we can look at how the idea of bringing materiality into the discussion helps to solve or to improve the problem. Orlikowski offers the concept of a ‘scaffold’ as the mechanism by which materiality can sustain knowledge. Scaffoldings have the following characteristics (Orlikowski 2006:461–2).

• Scaffolds are temporary: they are erected on a building site to support the construction of particular elements. They typically exist for the duration of
the project (or less) and are dismantled once the elements are completed or self-supporting.

- Scaffolds are *flexible*: they are constructed in situ and adapted to fit the particular local conditions. As such, they can be erected in many different situations.

- Scaffolds are *portable*: they are relatively quickly and easily assembled, modified and disassembled, as needed, on different building sites.

- Scaffolds are *diverse*: there are many different kinds of scaffolds—for example, scaffolds that allow people to walk along the outside of buildings, scaffolds that suspend workers from above, scaffolds that serve as structural columns to hold up slabs until the poured concrete is cured and scaffolds that serve as reinforcing formwork that then becomes integrated into the final element being built.

- Scaffolds are *heterogeneous*: they comprise multiple different components, reflecting both the requirements of the element(s) to be supported and the materials at hand.

- Scaffolds are *emergent*: they are erected over time, changing in form and function as needed to continue supporting the changing scale and scope of the element(s) being built. While in place, scaffolds afford a certain temporary *stability* to the disparate assembly of people, materials and space bound together.

- Scaffolds are *dangerous*: as temporary, emergent and rapidly constructed assemblages, they are vulnerable to breakdown and failure.

- Scaffolds are *generative*: they serve as the basis for other (creative) work, facilitating the performance of activities that would have been impractical without material augmentation.

- Scaffolds are *constitutive* of both human activity and outcomes, shaping the kind of construction work that is possible and the construction outcomes that emerge (for example, scaffolds afford the building of skyscrapers).

In situations in which knowledge has to be exchanged at a boundary—such as the one existing between developers and users—the main mechanisms facilitating the passage that could function as a scaffold are the boundary objects (Carlile 2002; Star and Griesemer 1989). Boundary objects are artefacts especially designed to ease the passage of knowledge. Boundary objects in ISD can be models, documents, diagrams, prototypes or software releases. During meetings, these boundary objects become the focal point around which discussion revolves.

The design of an effective boundary object is not as straightforward as it might seem. If badly designed, a boundary object can in fact impede the passage of knowledge. This is a known phenomenon in which the inappropriate use of a
boundary object strengthens the power position of one group or the other and
reinforces the boundary rather than bridges it (Wenger 2000). Let us consider
the characteristics of a boundary object applied, for example, to a possible
prototype created for the SteelCo case.

*The boundary object must be visual* (Brooks 1985; Carlile 2002). Visual artefacts
are easy to inspect and are quickly understood. The software prototype has to
replicate the environment of the users such that they can verify whether the
developer’s understanding of it is accurate enough. Visualisation responds to
the need for making knowledge explicit.

*The boundary object must be usable/functional* (Brown and Diguid 2001). Not
all knowledge can be made explicit by visualisation. Some knowledge that
remains tacit can be demonstrated only through action. By working with the
prototype, the users enact their daily routines and can immediately identify the
misunderstandings of the developers. Visual and functional boundary objects
will also facilitate the establishment of a common language (Wenger 2000).

*A boundary object must be up-to-date* (Carlile 2002). The prototype has to be the
newest product of the developers. The main function of the prototype is that the
developers can take home the comments of the users, change their understanding
of the problem and create more accurate solutions. If the developers present
to the users an obsolete prototype while they are already working on newer
versions two problems might happen. First, the developers will be focused on
newer problems and will miss the importance of the users’ feedback. Second,
invested knowledge will create inertia against the doing of rework.

*The boundary object must work both ways* (Boland and Tenkasi 1995). Prototypes
are built for the users to learn about the system’s possibilities but also for the
developers to collect feedback. Mechanisms must be built in the prototypes
to facilitate the collection of the feedback. For example, the prototype can be
made with a ‘recording’ mechanism built in so that everything done with the
prototype and everything said can be recorded and replayed at will. This feature
will give the developers a chance for retrospective sense making and facilitate
the improvement of the software.

Perhaps one other key characteristic boundary objects need to function, as
such, is that they have to be *incomplete*. Incompleteness generates the need for
concerted action.

We can imagine designing a special type of boundary object that supports
knowledge transfer by functioning as an imitator of the material world. Such
a boundary object should at the same time be temporary, flexible, portable,
diverse, heterogeneous, emergent, dangerous, generative, constitutive and
visual, usable, up-to-date, bi-directional and…incomplete.
As can be imagined, a boundary object lends itself very well to representing a material setting. For example, physical scale models or three-dimensional computer-aided design (CAD) drawings are normally used in construction (Gal et al. 2008). In the inventory case, there was so much ‘materiality’ defining the setting that the problem was not what to represent but rather comprehending the importance of this representation.

Example of use of materiality to scaffold knowledge exchange in the SteelCo case

In the initial phases of the SteelCo case, the developers began their investigation of the setting with the business problem, focusing on both the material and the conceptual sides of it. In this initial phase, the definition of the business problem was detailed, resulting in a material model of the system: a drawing of the inventory on paper and a conceptual software model. These models acted as boundary objects, at an early stage facilitating knowledge exchange from the users to the developers and enhancing the developers’ understanding of the domain.

The models became the basis for the design even though they were not directly related to how the software system was built (see more on this issue in Carugati 2008). The first version of the software—what was called the proto-prototype—presented to the users only a table of white numbers on a black DOS screen. From the lack of feedback from the users about this proto-prototype, it quickly became clear that these black-box prototypes would not further knowledge exchange, so a discussion was initiated to find a better way to do it. The discussion revolved around whether or not to include a visual simulator as an additional module in the inventory software. Supporters maintained that a visual simulator would improve understanding and feedback while their opponents replied that it was useless (the DOS screen respected the requirements) and too complex and time-consuming to develop. The two factions fought lengthy battles on the issue until a developer took out his computer and showed a visual simulator of a welding robot used at the shipyard. The visual simulator was showing on the screen the actual robot on the production floor. While the programmer was focusing on the numbers, the users in the background (they were literally standing behind the programmer) were commenting on their actual understanding of the situation:

- Hey, look there, that’s the turn where it always gets stuck…
- Yesterday in that corner they programmed it wrong so it welded a hole…
- Wow…programming it like that you could make it go much faster…
The visualisation had such an impact on the users that after the demonstration there was complete agreement that the visual simulator had to be done even though why it had to be done had different reasons for different roles. The materially grounded boundary object, used in a participative setting, had played a key role in bringing the different actors to agreement.

As a direct result of this discussion, the first real prototype integrated the DOS screen in a graphical user interface (GUI) that resembled the inventory seen from above. For this case, the discussion about the prototype was very limited whereas the second prototype received much more attention. The screen of the second prototype is shown in Figure 7.2. This GUI resembles the inventory seen from the top: the sea on the upper side, the two cranes with the same appellation as used in SteelCo and the grey plates. The left side of Figure 7.2 is what could be called a material software model. On the right side, we find instead the list of plates that have to be delivered on the day. This list comes from the database of the production plan.

On presentation of the first material prototype, the users were more relaxed than before in the discussion. They were smiling and definitely interacting in pointing out problems in the crane movements.

Unfortunately, the programmers presented this prototype after the planning algorithm was ready—as is evident from the central part of Figure 7.2 that specifies the executions for the cranes. As became clear later in the case, the material model should have been used before the planning algorithm was done. The time and knowledge invested in the creation of the algorithm caused the developers to resist the feedback of the users instead of welcoming it.

In the SteelCo case, the prototypes like the one in Figure 7.2 were good material models because they furthered understanding, but poor boundary objects because they impeded feedback and positive reconstruction of the work done. A characteristic that the prototype had lacked, as a boundary object, was usability. It was too complete and therefore not bi-directional because one group—the developers—was not interested in changing it.

Despite its poor result as a boundary object, the prototype as visualised in Figure 7.2 represented a turning point in the SteelCo case. It showed to all parties that progress had been made; it showed that there was a common point of understanding of the problems; and it showed this in a concrete and discussable way. As the project manager of SteelCo said, ‘It was as if the program was brought to life for us. It was not that the DOS version was wrong, maybe it was…I cannot judge that, but this one was different; we just got it.’
The programmers saw it in a similar way: 'The big change happened when we introduced the simulator; it was there that the discussion really started. We should have handled it differently...for the feedback...It would have been different if we had done it at the beginning but nobody had told us.'

These last two comments point back to Latour’s quote of rethinking the way we operate. The project manager and the programmers had considerable experience with software projects, yet for them the results of introducing the material model were surprising. In the discussion about the introduction of the simulator they were in fact against it. Given the results, the professionals’ attitude and their reactions, we can begin to delineate two design principles for IS development in complex settings

- whenever possible, start ISD efforts by developing a graphical simulator of the material environment of the users
- embed and use the simulator as a proper boundary object.
Use of materiality in ISD

In this section, we present some ideas on how to include the two design principles in ISD. Over the years, many strategies and techniques have been described that have improved greatly the chances of creating successful information systems. The difficulty in determining the business problem leads to the design of iterative processes that allow for frequent revisions. The need for knowledge exchange requires the creation of boundary objects as focus points in discussions. To reduce the negative effects of invested knowledge, short iterations and simple prototypes are required. This in turn reflects the need for splitting the software into smaller development tasks that can be tackled within one iteration. The combination of these techniques in different situations has given birth to the concept of agile methodology. To this we would add—responding to the practitioners’ call for light customer involvement and riding the wave of material knowledge proposed by Orlikowski—that starting a development effort with the creation of a simulator that mimics the material setting of the customer increases the amount of knowledge exchange while limiting the need for customer involvement.

A methodology that includes the design principles could involve the following steps

1. business problem definition
2. material modelling and usage modelling
3. iteration and release planning
4. design and implementation
5. evaluation
6. integration.

These steps can be viewed as the steps of an agile development or as a sequential process. It is, however, only through the iterative process and the integration of practice-based feedback that it can be used. If the agile paradigm is chosen then the methodology focuses on a process that facilitates frequent and constructive encounters between developers and users while at the same time trying to minimise the amount of time used for discussion without the support of the proper boundary object. The process involves repeating the phases of the methodology many times in an iterative fashion in order for the knowledge exchange to take place. The idea is to use the knowledge of the material setting as much as possible to minimise the impact of the development on the customer and instead to put effort in the activities of evaluation and validation, redefinition of use cases and iteration and release planning that are more meaningful when based on a materiality-based software prototype.
Figure 7.3 shows an example of how the methodology phases might unfold over a series of three iterations. For reasons of space, the stages of the methodology are indicated by numbers 1–6 in Figure 7.3. In the figure, the methodology stages are mapped against the system life-cycle activities proposed by the ISO 15704 to show which activities are involved in each stage. The figure shows that in the first iteration, when the material setting is discovered and modelled, few resources are used to define the business problem and then in the second iteration the business problem is revised as a result of learning from the first cycle. In stage four, the developers design, implement and test the system. In stage five, the users evaluate and test the system. In accordance with the characteristics of boundary objects, the prototype should be functional and therefore evaluation is a stage that involves operation of the system. The barred box in stage six represents the integrated system only for evaluation and testing purposes by the users. In the third iteration, stage six represents the introduction of the system in use. Integrating software whenever possible is not only a source of immediate value for the users; software releases are the best boundary objects since the users can provide feedback to the developers from real usage.

![Figure 7.3 An example of iterations over time](image)

The situation is explained further in Figure 7.4. This figure refers to the evolving nature of goals and objectives and shows a representation of the goals and objectives of the developers. At the beginning of the first iteration, the developers receive an initial understanding of the setting and they proceed to develop the materiality-based prototype (square one in the figure) as a boundary object. When they show the prototype to the users, they (together) achieve a new understanding of the goals. They will easily find out if a part of the first prototype was out of scope because of its visible and familiar...
characteristics. In the second iteration, the developers proceed to develop the second prototype and correct the first, represented by the area of overlap of the squares marked with 1 and 2. In the following presentation, the situation is repeated. Parts of the second prototype are out of scope and parts have to be corrected. The process continues in this fashion until the work done during the entire development effort covers all the needs for the final version of the goals and objectives. Obviously, the shape of goals and objectives changes not only because the software prototypes help the users to understand what they want from the IS, but because something in the environment has changed.

Figure 7.4 Evolving goals, objectives and development efforts

Figure 7.5 depicts what this methodology could look like in a case similar to the SteelCo case in which the scope is to develop planning software. The figure shows an example of six iterations. In the beginning, a first version of the requirements for the IS is formulated based on the initial definition of the business goals. In the first iteration, the focus is on creating the simulator of the material setting that must be used as a boundary object to focus the discussion between developers and users. The setting is the first indicator of what the client needs to do. As a surrogate of complex and time-consuming interaction with the customer, the setting can be observed, studied, photographed, and so on, and it can be used as the basis for creating the material model. The final system might lose the graphical characteristics, but the initial attempt has to be grounded in materiality as much as possible. In the first iteration, a GUI representing the setting is created along with a database connection (DB). This is to prepare, at the very beginning, a visual and usable prototype. In this phase, there is no intelligence in the system and all the operations are left to the user (the boundary object has to be incomplete and the scaffold emergent). In the following iterations, more core elements are added as a result of agreement between users and developers.
Figure 7.5 ISD process for information systems (for planning)

Generic system architecture and material model

Regarding the question of how the material model has to be embedded in the system, Figure 7.6 depicts a generalised version of the system architecture used in the SteelCo case. The material model, scaffolded by the materiality of the setting (reality in Figure 7.6), was the visible part of the software and, while interacting with the users on the human side, it provided the required data to the data model instead of the reality. This was particularly convenient in the SteelCo case because the system was for planning so the material model in fact worked as a simulator for the system core to test new plans. This could be useful in many settings where, for example, there is the need to test new routines. This could be done conveniently without disrupting the work environment. Figure 7.6 summarises the characteristics of reality as a scaffold of the material model as well as stating the characteristics of the material model if we want it to work as a boundary object. By paying attention to both sides of the figure, and recalling that they are by no means stable, we can create software architectures that are fit to support emerging knowledge in an emergent and changing world.

Figure 7.6 Generic system architecture and material model
Conclusions

In this chapter, we have proposed two design principles to account for and take advantage of material knowing in ISD. The principles are

- whenever possible, start ISD efforts by developing a graphical simulator of the material environment of the users
- embed and use the simulator as a proper boundary object.

These design principles help the ISD process in different ways: helping to make software tangible—a problem not felt as much in other fields but very important in ISD; enhancing knowledge exchange and quality of feedback; and decreasing the need for user participation in ISD.

The point of departure of our chapter is that, still today, ISD methodologies fail to meet the requirements of software houses either because the customer cannot be involved as much as they should or because the technology is too complex to allow for rapid solutions.

We have, therefore, proposed a design principle that uses the environment instead of people to aid the process. It starts from the materiality of the users’ setting to achieve the knowledge exchange necessary without the continuous involvement of the customer.

Materiality of the setting presents high potential for improving the practice of ISD because of its nature as a boundary object and because of its connection to what we know and how we learn. On the other side of the coin, it also represents a highly underdeveloped element in ISD and as such it needs more exploration. This study presents an attempt at taking materiality seriously in ISD and also represents a call for more studies into this field.

References


Markus, M. L. and Keil, M. 1994, ‘If we build it, they will come: designing information systems that users want to use’, *Sloan Management Review*, vol. 35 (Summer), pp. 11–25.


