Workstyles: An Evaluation Model for the Design of Interactive Systems

James Wu and T.C. Nicholas Graham
Department of Computing and Information Science, Queen's University
Kingston, Ontario, Canada, K7L 3N6
{wuj,graham}@cs.queensu.ca

Abstract: This paper presents the *workstyle model*, a novel technique for recording the working style of people using an interactive system. Workstyle complements task modeling by providing information on how people communicate and coordinate their activities, and by showing what style of artifact the work is to produce. We have applied the workstyle model to the evaluation of UML design tools and the design of a new tool, the Software Design Board. The design of the model itself was informed by studies of tools and designers both at Queen's University and at a large software development organization.

Key words: Model-based design, work style, software design tools, UML

1. Introduction

Task models (Diaper, 1989) help in understanding the goals and activities of users of a software system. Through this understanding, the designer of a system can ensure that users' tasks are supported. Techniques such as cognitive walkthrough (Wharton et al., 1994), task simulation and model checking (Paternò and Santoro, 2000) allow user interface designs to be checked versus task models, exposing areas where the system fails to adequately support user tasks, or where user tasks are supported in an inconvenient manner. Task models therefore help to ensure that software systems meet users' needs.

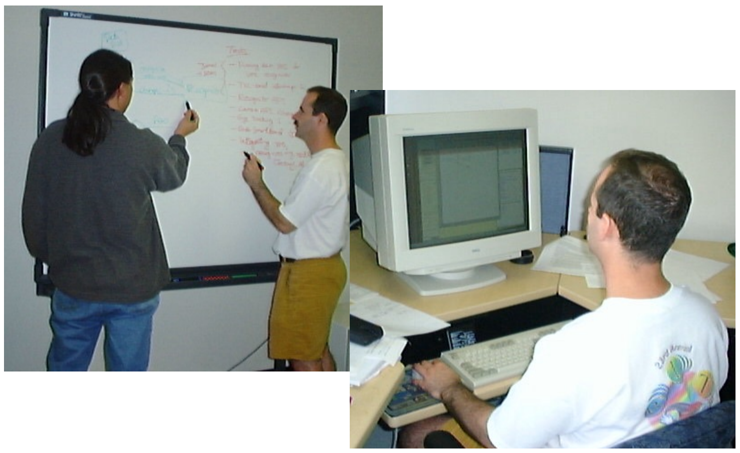

Figure 1. *Different tools (e.g. a whiteboard and a PC) may support the same tasks, while affording different styles of work.*

Task analysis techniques such as *GOMS* (Card et al., 1983) define user tasks in terms of goals, operations, methods and selection rules with the intention of providing an 'executable' algorithmic description of the task. The *User Action Notation* (UAN) (Hartson et al., 1990) and *ConcurTaskTrees* (CTT) (Paternò et al., 1997) also provide notations for tasks defined at this level. These techniques define user's goals at the lowest level – for example a GOMS or UAN goal could be "Delete a file". They provide a specification for the design of users' interaction with the system. UAN and CTT can also be used to encode hierarchical task models in the classical sense (Annett and Duncan, 1967), approaching the task analysis at a much higher level. These methods consider goals such as "Maintain a radar system while in flight", and redefine them in terms of lower level operations, such as "Adjust radar". Eventually, these high-level (or *user* tasks) are refined to lower level tasks that describe how the user's goals are realized with a specific system. *Task-Knowledge Structures* (TKS) (Johnson et al., 1988) helps in the analysis of user tasks, considering tasks to be concepts that have specific representations in the human mind. TKS structures are intended to represent the different aspects of knowledge that a user possesses in such a way as to provide a basis for lower-level task analysis techniques. The assumption that users' tasks have internal structure facilitates the prediction of how the user will carry out the tasks, as well as how various aspects of task knowledge interrelate.

While these task-modeling techniques are all intended to ensure that a system sufficiently supports the user in his or her task, they are less successful in capturing what it means for a task to be supported in a usable manner. To address this problem, we present the *Workstyle Model,* a novel technique for capturing some aspects of users' preferred style of work. The Workstyle Model complements task modeling by providing information on how people communicate and coordinate their activities, and by showing what style of artifact the work is to produce. This allows analysis of whether an interactive system design supports users in performing their tasks, and of whether this support is consistent with the users' working style. The model concentrates on users' preferred collaboration styles and on the desired properties of artifacts developed through this collaboration.

The Workstyle Model was developed in the context of the Software Design Board project, a project aiming to provide better tools for software design. The model has been validated through evaluation of existing design tools, and has motivated the design of a new software design tool.

The paper is organized as follows. In sections 2 and 3, we motivate and introduce the model. In section 4, we present the Software Design Board as a case study of the use of the model. The Software Design Board is an electronic whiteboard-based tool supporting the collaborative creation of design diagrams in the Unified Modeling Language (UML) (Rumbaugh et al., 1999).

2. Motivation

Later in the paper, we present the case study of a software design tool called the *Software Design Board (SDB).* The SDB aids in the implementation design of software systems. The Workstyle model was applied in the external design (user interface) of the SDB. To motivate the use of the Workstyle model, let us first consider the tasks a software designer performs:

- *Design a software system:* Here, the goal is to create a strategy for how the system is to be implemented. This may involve architectural design, design of protocols, selection of off-the-shelf components, data modeling, etc.
- *Convey design to other team members:* Ensure that other team members understand how the system is to be built. This typically involves the creation of design documents, and organizing presentations and meetings.
- *Split development project into work packages:* Here, the different components from which the system is composed can be given to team members, allowing parallel work. The design has to be sufficiently clear at this point that the interface of each component is well understood.
- *Analyze properties of the design:* Ensure that the design satisfies required properties such as availability, security and performance.

Central to each of these tasks is the creation of *design artifacts*. A design artifact records some aspect of a design, such as conceptual architecture, interface to a particular subsystem, design of a key data structure, etc. In addition to natural language, numerous notations exist for recording design, including the popular UML notation.

A system task model can help us understand the activities involved in creating a UML diagram: drawing and labeling nodes, connecting them with relations, editing and reformatting diagram elements, and so forth. Such a model of design activities might lead us to develop a tool similar to Rational Rose, permitting mouse-based structural editing of design diagrams.

However, before committing to such a design, it is important to understand the users' preferred workstyle in addition to understanding the tasks they need to perform. For example, designers may be working in a brainstorming style, or may be recording precise documentation from which the system is to be built. As illustrated by figure 1, the brainstorming style is well supported by a whiteboard, while the precise design style is well supported by a traditional Computer-Aided Software Engineering (CASE) tool. Both tools support the activities identified through the task model. However, they support the tasks in different ways, appropriate to either the brainstorming or precise design styles of work. We can summarize the properties of these workstyles as follows:

- *Brainstorming Workstyle:* Initial design is typically a brainstorming activity, often involving collaboration. Artifacts are typically created using informal media such as paper or whiteboard. There is little requirement for precision or correctness; e.g., requiring people to adhere to the precise syntax of UML would hinder the flow of ideas. Brainstorming normally results in the fleshing out of a coarse-level design, where details will be pinned down later. Brainstorming is characterized by rapid interaction, and use of social protocols to mediate turn-taking.
- *Precise Design Workstyle:* Precise design must refine details to a level that is suitable for analysis and distribution into work packages. In precise design, adherence to the design formalism becomes more important, as vagueness in the meaning of the design can lead to analysis errors and integration problems. Precise design may be performed with the aid of a software design tool such as Rational Rose (Quatrani, 1998). Precise design typically involves coarse-grained collaboration only, where the design task is split amongst individuals, who may occasionally meet for design reviews or further brainstorming-style discussion.

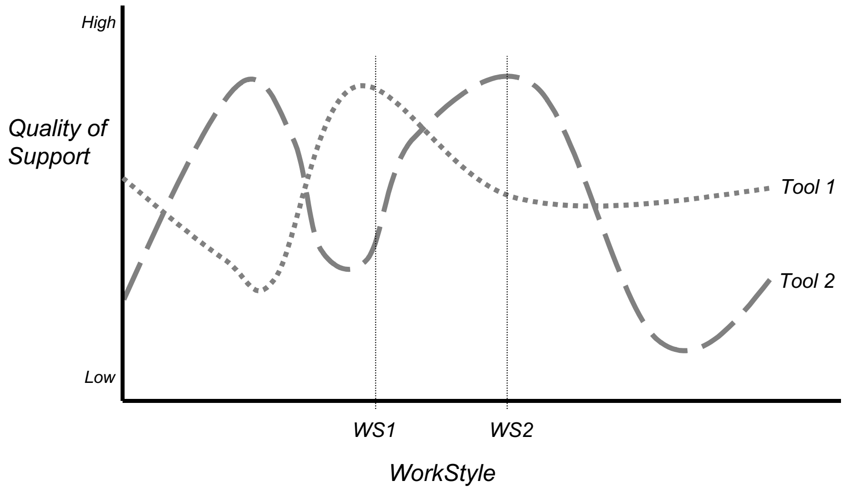

Figure 2. *Tools may provide differing levels of support for different workstyles. Here, tool 1 better supports workstyle 1, while tool 2 provides better support for workstyle 2.*

As we shall see in the following section, the workstyle model helps in the analysis of peoples' goals and tasks by helping to understand the style of work appropriate to carrying out the task. This in turn helps in the design of interactive systems, by providing a more complete understanding of the tasks being performed.

3. The Workstyle Model

The goal of the Workstyle Model is to provide a mechanism for recording styles of work. This complements task modeling by helping to expose the context in which tasks are being carried out. The Workstyle Model helps in:

- *Design of interactive systems:* In addition to considering what tasks are to be performed, it is important to understand the context (or style of work) in which the tasks are carried out. E.g., a tool supporting the brainstorming workstyle might be designed significantly differently from a tool supporting the precise design workstyle.
- *Tool adoption:* When considering purchase of a software tool, it is not sufficient to examine whether the tool supports the tasks that users need to perform. Mismatches can be identified between the style(s) of work supported by the tool and those most natural to the task.

Figure 2 shows how workstyles relate to tools. A particular tool will be suited better to some workstyles than to others. In this example graph, tool 1 provides better support for workstyle 1 than tool 2, and vice versa. In designing or choosing a tool, it is therefore important to consider workstyle.

Workstyles also help understand the role of time in tool usage. As a task progresses, users' workstyles may change. As shown in figure 2, moving from (e.g.) workstyle 1 to workstyle 2 may render an initial choice of tool 1 inappropriate. This requires users to continue with tool 1 despite its poor match with the new workstyle 2, or to change to tool 2. Tools should be designed with low *impedance*, allowing users to easily move between them.

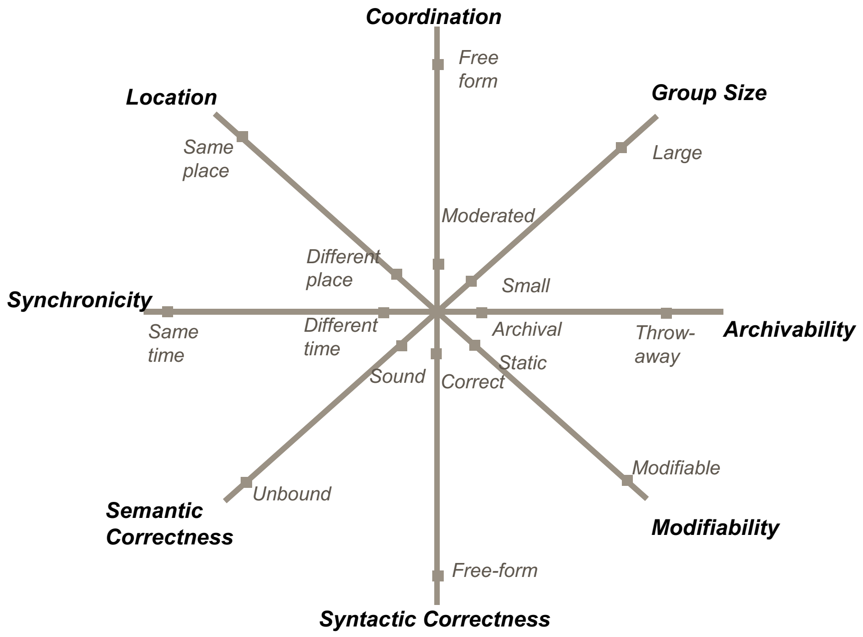

Figure 3. *The workstyle model presents an eight-dimensional space on which working styles can be plotted. Workstyles are decomposed into collaboration style, and style of artifact being produced.*

Finally, users' physical context may impose aspects of their workstyle. For example, if users are physically separated, some kinds of collaborative workstyles may be difficult to support.

The following sections introduce the notation used to express workstyles. Section 4 then uses the case study of the software design board to illustrate how the workstyle model may be applied.

3.1 A Notation for Describing Workstyles

We characterize the workstyle of a group in terms of how the group chooses to collaborate, as well as the kinds of artifacts the group chooses to create. As shown in figure 3, the workstyle model characterizes working style as a space of eight continuous dimensions. The first four describe collaboration style; the remaining four describe the properties of artifacts being created. A particular workstyle can then be represented as a point in this space. A particular tool may support a set of workstyles, which can be represented as a region in this space. As will be shown in section 4, it is then possible to compare peoples' desired workstyles to those supported by tools, helping to identify potential tool mismatches.

3.2 Dimensions Describing Collaboration Style

The first four dimensions describe how people collaborate while performing their tasks. Any particular choice along a dimension is influenced not only by personal preferences, but tool availability, and other situational factors. For example, while traveling, a designer may have no other options than telephone or email by which to communicate. Similarly, choice of a particular tool may limit the available means for communication. The dimensions that describe collaboration style are:

- **Location:** The location axis refers to the location of the people involved in the collaboration. People may be in the same place (*co-located*) or in different places (*distributed*.) In general, collaboration becomes more difficult at a distance (Seaman and Basili, 1997), and requires more support from tools. For example, artifact repositories found in many software design tools support remote collaboration by providing distributed access to shared objects. However, such systems do not facilitate distributed collaboration beyond support for asynchronous, artifact-level communication.
- **Synchronicity:** The synchronicity axis captures temporal aspects of the collaboration. People may work together at the same time (*synchronously*) or at different times (*asynchronously*). Face-to-face or telephone conversations are examples of same-time interaction; email conversations or information sharing through a Lotus Notes database are examples of different-time interaction, while a rapid exchange of emails falls between the two.
- **Group Size:** The group size axis describes the number of people that may be involved in the collaboration. Group size influences what styles of interaction are practical. For example, brainstorming may work in small groups, whereas larger groups require support of tools or communication processes. Asynchronous collaboration tools typically support larger group sizes than tools that support synchronous collaboration.
- **Coordination:** This axis represents the model used to coordinate (Malone and Crowston, 1990) the group's activities. For example, in a brainstorming session, free-form coordination is typical. Social protocols can determine the order of speaking or modifying shared artifacts. In meeting situations, more rigid coordination is typical, relying on a chair or formal rules of order. Asynchronous tools typically rely on moderated coordination, such as check-in/check-out protocols, while synchronous tools like telephone or chat support more free-form interaction.

3.3 Dimensions Describing Artifact Style

The remaining four dimensions describe the properties of artifacts that result from the users' tasks. As before, a particular choice along a dimension is influenced by factors such as personal preference and tool availability, as well as the collaborative context, as described above. For example, a software design tool may support a large group size across a large distance with asynchronous communication and a rigid coordination model. However, it may not provide flexibility with respect the syntax and semantics, requiring conformance to a particular formal language. The properties describing artifact style are:

- **Syntactic Correctness:** This axis represents the degree to which the production of an artifact is required to follow a precise syntax. For example, a programmer must follow the rules of the programming language being used, while a designer may choose to follow the precise rules of a notation such as UML. However, in the early stages of design, a requirement to adhere to precise syntax may hinder creativity by diverting designers from the global concepts of design (Tang, 1991; Landay and Myers, 1995).

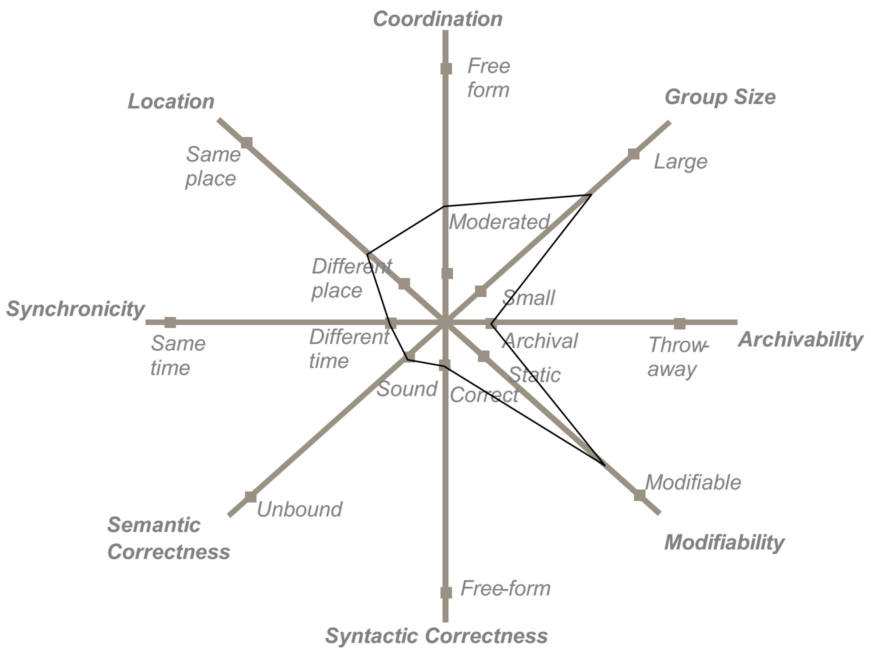

Figure 4. *Workstyle supported by Rosetta is similar to many popular UML design tools, differing only along the Coordination axis. Rosetta supports lighter-weight coordination than many UML design tools.*

- **Semantic Correctness:** This axis represents the clarity of meaning of produced artifacts: that is, the degree to which they may be ambiguous or contradictory. The production of semantically sound artifacts may be impractical and unnecessary. For example, design documents often contain contradictions, inconsistencies and missing information, particularly in the early stages of the design process. Humans can often work effectively despite the presence of such contradiction (Finkelstein et al., 1994).
- **Archivability:** Archivability represents the ease of storing an artifact so that it can be used at a later time. For example, word processing documents have high archivability, as they can be saved to disk and retrieved later. A standard whiteboard has poor archivability - once it is erased, its contents are lost. Archiving is considered more difficult if an archived version cannot be used in the same way as the original. For example, a whiteboard drawing can be archived by photographing it; however, the photograph can no longer be manipulated on the whiteboard.
- **Modifiability:** This axis represents the ease with which an artifact can be modified. Modifiability is task-dependent. For example, small modifications to a standard whiteboard drawing are simply performed by erasing and redrawing. Similar modifications in structured drawing editors often require a sequence of commands. A modification such as reformatting a complex diagram on a whiteboard, however, is difficult. Large-scale modifications to a diagram produced using a drawing program are more completely supported, but may require complex editing operations.

4. Application of the Model

The Workstyle model can be applied to both the evaluation and design of interactive design tools. The model has been applied to the development of a UML design tool, the Software Design Board, as well as to evaluate the strengths and weaknesses of a variety of Computer-Aided Software Engineering (CASE) tools. In explaining the application of the model, we will use the example of the design of the SDB.

In designing the SDB, we first used the model to show how existing UML design tools fail to match the workstyle of software developers. This was performed as part of a six-week study of developers in a large software development organization. We then used the model to help motivate the design of the new tool. In presenting the case study, we will therefore first consider the workstyles afforded by existing software design tools, before analyzing the workstyle of the Software Design Board.

4.1 The Rosetta Software Design Tool

It is useful to consider the workstyle supported by popular UML design tools such as Rational Rose, Aonix Software through Pictures (STP) and Together Control Center. Workstyle plots for these tools are similar to figure 4. As can be seen, these tools support small-to-medium groups. The activities of different designers are coordinated through a repository, using locking and merging to support concurrent work. Designers can work in different places. Communication is different-time, through the contents of the repository. The tools provide structure editing that guarantees that designs will be syntactically correct. This in turn helps designers ensure that designs are semantically sound. Since the tools provide structure editing and the ability to save and restore designs, designs are highly modifiable and easily archived.

Rosetta is a lightweight tool for designing, documenting and checking Java programs (Graham et al., 1999). We present it here to demonstrate the application of the Workstyle model to the evaluation of a typical software design tool. Though Rosetta differs in details from tools such as Rose, it stands as a suitable example of the genre of tools to which these tools belong. The workstyle plot for Rosetta is shown in figure 4.

As an academic tool, Rosetta is restricted to a small subset of UML. Similarly, it does not support code generation, or reverse/round-trip engineering. Instead, it provides a conformance checker to identify deviations between designs and corresponding code. This tool also facilitates HTML documentation of designs and corresponding Java code. The research goals behind this tool were to provide a lightweight, process-neutral tool, whose use did not impose heavy cognitive overhead on designers. Rosetta's evaluation along each workstyle dimension is described below.

- *Archivability:* Rosetta provides a distributed repository to support archivability, allowing designs to be accessed anywhere over the web. The tool also supports XMI, and can export its models as XML documents, facilitating interchange with other tools. Therefore, Rosetta provides high archivability.
- *Modifiability:* The user interface provides a structured editor sufficient to fully maintain all diagram types supported by the tool. Therefore, Rosetta provides high modifiability.

- *Syntactic Correctness:* Rosetta uses a formally defined syntax that can be seen as a simple subset of UML, and is defined through relational algebra. The tool supports the creation of both object and class diagrams, and the editor provides all the necessary primitives for creating such designs. Furthermore, the editor does not allow syntactically incorrect diagrams to be drawn. This means that designers are constrained to the limited syntax provided by the tool. Therefore, Rosetta insists on high syntactic correctness.
- *Semantic Correctness:* The Rosetta design notation has a precisely defined semantics based on the relational algebra through which it expressed. Every design has a precise meaning, which facilitates automatic checking of conformance between design and code. The *Conformance Checker* tool reports conformance errors between design and code. Rosetta does not require conformance errors to be immediately repaired, allowing designs to contain contradictions, inconsistencies and missing information. Therefore, Rosetta permits designs to be semantically unbound.
- *Location:* Rosetta supports distance collaboration through its web-based architecture. The tool is intended to be accessible from any Java enabled web browser, and requires no further installation or configuration. Rosetta does not provide any additional functionality, such as person-to-person communication mechanisms, to further facilitate distance collaboration.
- *Synchronicity:* Rosetta is accessible from any point on the web. The repository can therefore be accessed by a group of designers, and supporting asynchronous collaboration. The tool allows parallel development of a model by decomposing the project into a collection of diagrams. Diagrams cannot be accessed concurrently. Therefore Rosetta supports asynchronous collaboration only.
- *Coordination:* Rosetta imposes light-weight coordination on developers' activities. At the level of diagram, coordination is imposed by "check out" functionality of the concurrency control mechanism. Beyond this level, no coordination is imposed on interaction between collaborating developers. Furthermore, the tool does not impose any process coordination. The tool is not linked to any particular code development environment. Additionally, diagrams and code are not directly linked, and may be developed in any order. There is no requirement that code conform to corresponding designs; however, conformance can be checked via the *Conformance Checker*. This reduction in coordination can serve to facilitate creative/experimental design.
- *Group Size:* Rosetta can support a reasonably large group of developers working asynchronously. The limit on group size is imposed by the size of the project, i.e. the number of diagrams, as well as the locking granularity provided by the tool. Rosetta does not directly provide any further functionality to facilitate communication and collaboration in a large group.

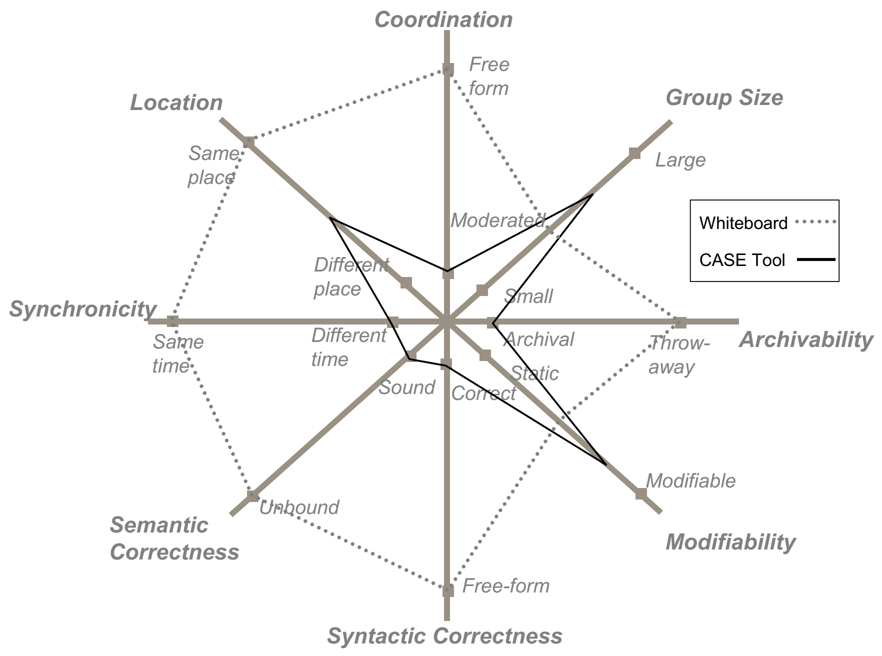

Figure 5. *Ideal workstyle for brainstorming a software design overlaid with workstyle supported by existing UML CASE design tools*

4.2 Tools for UML Design

UML design tools such as the Rosetta tool described above are a good fit with workstyles where relatively little communication with other designers is required, and where the goal is to create precise, archival designs. The tools are particularly helpful in enforcing UML syntactic and semantic rules, and provide good support for editing and archiving documents.

However, as discussed in section 2, traditional UML design tools provide poor support for the early stages of design, such as brainstorming. During these phases, designers spend large percentages of their time on communication tasks. For example, two studies show communication activities as requiring between 70% (DeMarco, 1987) and 85% (Jones, 1986) of software developers' time.

This communication is largely informal in nature: Kraut and Streeter (1995) report that software developers report "discussion with peers" as their most important method of coordinating their activities. Following a large study of a software development project in the U.S. Navy, Chmura and Norcio report that informal discussion is correlated with progress in design, and that the more complex the design problem, the larger the percentage of designers' time is spent on discussion (Chmura and Norcio, 1986). Studies of design tasks in general have shown that in brainstorming sessions, the majority of time is spent talking and gesturing at the artifact being produced (Bly, 1998; Tang, 1991). In simple activity analysis of brainstorming tasks, we have observed that as little as 5% of brainstorming time is spent actually producing the design artifact, while the remaining 95% is spent discussing it.

At the same time, it has been observed that traditional design tools provide poor support for informal communication (Vessey and Sravandapudi, 1995), and this lack of support has been linked to low adoption rates of these tools (Iivari, 1996; Jarzabek and Huang, 1998). In studies of designers in a large software company, we observed that the use of UML design tools was low, and that much design was performed with informal media such as paper or whiteboards.

The inappropriateness of UML design tools (such as Rosetta) for early stages of design can be clearly shown by examining the brainstorming workstyle. As shown in figure 5, brainstorming is typically carried out by small groups working face-to-face. These groups typically use free-form coordination, using social protocols to determine who gets to speak or write next. In brainstorming, designers do not wish to be distracted by requirements to be syntactically correct, or even semantically sound (Bly, 1988; Tang, 1991). Modifiability is important as early designs evolve rapidly. Archivability is important to allow early designs to be migrated to more formal designs.

Figure 5 shows that while UML design tools support the core tasks of the early stages of design, they do not support the workstyle of early design. The emphasis on asynchronous, moderated work with strong emphasis on syntactic correctness and semantic soundness is thoroughly incompatible with the free-form brainstorming workstyle. Ivarii's study shows that once designers move beyond this early design stage, they tend not to record the designs in a CASE tool unless management requires it. We therefore believe that figure 5 demonstrates a large part of the problem with current UML design tools.

4.3 The Software Design Board

To address these problems, we have developed the Software Design Board, a prototype UML design tool that supports a variety of workstyles appropriate to the early stages of design. As can be seen in figure 1, the software design board is physically based on an electronic whiteboard. This whiteboard is a touch-sensitive membrane allowing drawings made with a stylus to be captured. These drawings are then projected onto the whiteboard display using a projector.

Using the Software Design Board, designers can create UML designs simply by drawing them, similarly to the free-style drawing of the Tivoli system (Pederson et al., 1993). This gives the designer the full interaction flexibility of media such as paper or standard whiteboards. This keeps interaction with the tool at an informal level appropriate to early stages of design. This contrasts with other whiteboard-based UML tools such as Knight (Damm et al., 2000), which is a structure-editor that uses a proprietary gesture language to specify editing operations. The user interface builds on standard whiteboard interactions by implementing a massive work area that is both zoomable and scrollable. This provides sufficient space for all concurrent work to be maintained without opening and closing documents as the focus of work changes, or sacrificing existing work for free space. Work that is not of current interest may be zoomed and scrolled, while work that is the current focus can be magnified and centered. Once designs

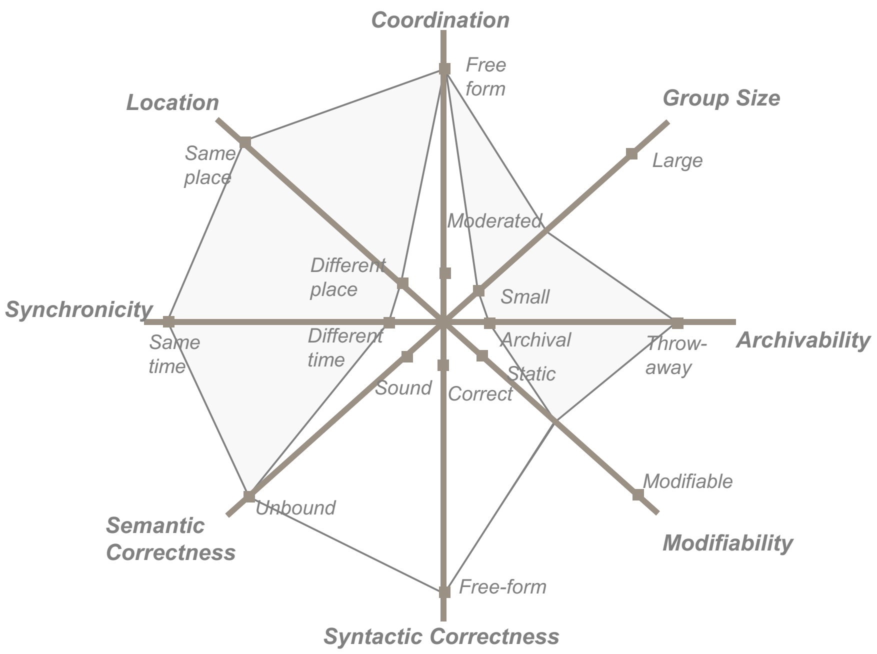

Figure 6. *Workstyles supported by the Software Design Board. These range over same-place and different-place, same-time and different time, small to medium groups. Consistently with early stages of design, coordination is free-form, with no restrictions on syntactic correctness or semantic soundness. Designs are somewhat modifiable through standard whiteboard erasing and redrawing. Designs can be archived by exporting them in XML to a standard UML CASE tool.*

have been drawn, the strokes may be passed to an experimental UML recognizer tool (Lank et al., 2001) that translates the scribbles into structured drawings, reformats them, and exports them in an XML format suitable for use in any XML compliant UML CASE tool. Such a tool may be used to further analyze and evolve the design in ways not supported by Software Design Board. The recognition of the hand drawn strokes is interactive, allowing mis-recognized items to be corrected during the recognition process.

The Software Design Board supports collaborative work in a variety of ways. Small groups can work together at the electronic whiteboard, similarly to working with a standard whiteboard. Different-time work is supported (as with a traditional whiteboard), by leaving board contents available for others to look at. Additionally, the drawing area of the board can be divided into arbitrarily sized segments that may be flexibly shared among distributed collaborators. Different segments can synchronously share different design artifacts with different groups of people. The current implementation only permits free-hand stroke data to be shared in this manner, though future implementations will allow sharing of entire applications through this mechanism. Additionally, shared segments are persistent, allowing fluid transitions between same-time and different-time collaboration. Different-place work is also supported by a variety of communication mechanisms. Collaborators sharing artifacts through a shared segment may maintain a voice connection. Additionally, gestural information (pointing, circling, etc.) is captured and passed between shared segments, as motivated by the VideoWhiteboard (Tang and Minneman, 1991). Telepointers allow physical gestures to be used to indicate interest and focus of attention. This provides an intimacy of interaction and a sense of contextual awareness that is typically available in face-to-face interaction.

Since not all users may have electronic whiteboards available, the Software Design Board can also be hosted on a standard PC. The interface is plastic, providing UI mechanisms optimized for the hardware on which it is running (whiteboard or PC). The PC interface is mouse-based, and relies on drop-down menus and tool bars for control of the application. Gestural information is communicated using the mouse. On the electronic whiteboard, the interaction is stylus-based, and relies on pie-menus and pen gestures to control the application. To capture physical gestures, a camera is used to track the stylus, and computer vision techniques are used to determine its coordinates within the shared segment. This allows gestural information to be shared even when the stylus is not touching the whiteboard.

4.3.1 Evaluating the Software Design Board

The Software Design Board's evaluation along each workstyle dimension is described in more detail below.

- *Archivability:* The Software Design Board does not integrate any kind of repository or revision control system to facilitate archivability, though nothing prevents design artifacts from being maintained in a third party tool. Similarly, whiteboard content can be imported into any OLE[1] enabled tool to further enhance archivability. The UML recognition tool supports XMI, and can export its models as XML documents to facilitate interchange with other tools. Like Rosetta, the Software Design Board uses a proprietary UML model.
- *Modifiability:* The user interface is a free-form drawing tool that currently supports only creation and deletion of free-hand scribbles. It does not support artifact modifiability like a standard whiteboard. Scribbles can be deleted only in their entirety, not partially erased then altered as on paper or standard whiteboards. However, as the nature of the application is such that each scribble can be treated as a unique element, full structured editing is possible in future implementations.
- *Syntactic Correctness:* Software Design Board does not enforce any particular syntax – freehand drawings may consist of any and all manner of strokes. However, the UML recognizer implements a specific subset of UML, and only recognizes hand drawn diagrams containing elements of this subset. Thus, syntactic correctness is required for the recognition process, though never enforced while creating the free-hand design.
- *Semantic Correctness:* Freehand drawings made in Software Design Board may have arbitrary and varying semantics. There is no requirement that the drawings be unambiguous or free of contradiction. However, application of the UML recognizer to the drawing encourages semantic refinement, though still does not enforce correctness.
- *Location:* Both co-located and distributed collaboration styles are supported by this tool. The whiteboard-based nature of the tool easily supports same-place

[1] Microsoft's *Object Linking and Embedding*

interactions, while shared segments, gestures, and voice connections allow the artifact and communication spaces to be shared between remote locations.

- *Synchronicity:* The Software Design Board supports both synchronous (through shared artifacts) and asynchronous (persistent artifacts) interactions. Designs may be developed in parallel on separate boards, merged while working synchronously, then separated again for further parallel development.
- *Coordination:* The tool does not impose any coordination model on interaction or in the design process. Designers may flexibly share artifacts without restriction, beyond permission of the original author. Additionally, there is no built in design process, nor any requirement that designs be synchronized with code. However, such functionality is available by exporting designs to appropriate CASE tools.
- *Group Size:* Software Design Board supports a large group of developers, with no real upper limit to group size. Segments can be shared with unlimited collaborators, within the capabilities of the underlying network, and there is no limit to the number of segments that may be created and shared. Similarly, the working space of the tool is sufficiently large as to never practically limit the number of people collaborating.

4.3.2 Transition Between Workstyles

As discussed in section 3, the workstyle model helps show where transitions between workstyles might lead to problems of tool *impedance*. Tool impedance refers to cases where a tool is no longer appropriate to the current workstyle, but where it is difficult to change to a more appropriate tool.

In addition to addressing a markedly different workstyle to typical CASE tools, the Software Design Board supports low-impedance transitions between a number of different workstyles:

- *Same-place to different-place interaction:* designers may share a single whiteboard, or dynamically connect to remote whiteboards or PC's. The SDB design recognizes that workstyles may be imposed by context rather than chosen by users. E.g., a mobile user may not have network connectivity, and therefore need work asynchronously. Later when a network is available, the user can then connect for synchronous interaction.
- *Free-form to moderated coordination:* by exporting the design to a CASE tool, designers may take advantage of design repositories with check-in/check-out functionality. Exported designs can also take advantage of superior archivability and modifiability provided by CASE tools.
- *Syntactic/semantic freedom to syntactic/semantic correctness:* applying the UML recognizer requires the design to be made syntactically correct, and encourages semantic refinement.
- *Same-time to different-time interaction:* once a same-time brainstorming session is complete, designs may be exported to a CASE tool for asynchronous access, or may be left on the whiteboard as a reference.

4.3.3 Implementation Status of Software Design Board

The Software Design Board is currently implemented as a collection of tools, providing the functionality described above. Ongoing work is integrating these tools into a unified framework.

The core of the SDB is a drawing tool based on an electronic whiteboard. The drawing tool supports gesture recognition, allowing hand-drawing of structured diagrams. The tool is designed for stylus input, integrating gesture-based editing, zooming, menus, etc. The SDB uses the CALI gesture recognition system (Fonseca and Jorge, 2000) and the Pie menu system (Hopkins, 1991). The SDB may nevertheless be run on a standard PC, on which traditional interaction techniques are supported.

Any OLE-compliant application may be used within the SDB, such as PowerPoint and Word.

Drawings may be divided into segments, which can be shared with other users. Arbitrary sharing topologies are possible, where one segment can be shared by multiple users, and multiple shared segments can shared by different groups. Sharing is currently limited to SDB drawings. Work is ongoing to provide sharing of external applications, using techniques similar to those of the JAMM system (Begole et al., 1997).

Conversational gestures are captured by a camera. Currently this uses the built-in tracking of a Sony AF CCD camera (Wu et al., 2002). We are currently implementing a more accurate and less expensive technique based on a pen which emits infrared light to provide positioning information. This pen can be tracked by an inexpensive camera with infrared filter.

The UML recognizer (developed by Lank et al., 2001) is currently a standalone application. This will be integrated with the rest of the SDB. The UML recognizer is robust, but does not yet have sufficient accuracy for production use.

5. Analysis

The preceding section has illustrated the key uses of the Workstyle Model: analyzing problems with tool adoption and helping in the design of new tools.

Analyzing Problems with Tool Adoption: The Workstyle Model helps to show how well an existing tool will meet the needs of its users. Supporting users' tasks is not sufficient; a candidate tool must also support users' preferred workstyle.

To apply the model, it is necessary to first determine the range of users' workstyles. This is then compared to the workstyles supported by the tool (e.g., figure 6). If the tool's workstyles poorly match the users' preferred workstyles, the tool is potentially unsuitable for adoption.

In the example of section 4, we saw that early design is characterized by a brainstorming workstyle. Most software design tools, however, support a precise design style. Comparing the workstyle plots of existing tools versus desired workstyle clearly illustrates the incompatibility of existing tools with early design of software. As shown in figure 6, differences range over coordination model, support for collaboration, and requirements to adhere to formal syntax of the UML design notation. Designers therefore typically choose informal media such as whiteboards or paper for initial stages of design.

As designers move from initial design to precise design, it becomes more desirable to move to a traditional CASE tool, as the desired workstyle approaches that provided by such tools. However, there is high impedance in moving data from hand-drawn diagrams into a CASE tool – typically the diagrams must be completely re-done. The fact that CASE tools are incompatible with designers' initial workstyle, and that there is high impedance to moving to them, hinder their adoption.

The workstyle model helps in understanding the range of workstyles a tool can support, clarifying at what points users will be required to work in an inconvenient style or change tool.

Helping in the Design of New Tools: Modeling users' preferred workstyles helps in the design of tools supporting those users' tasks. Models can show how users ideal want to work. This can be a range of workstyles (therefore a region in a workstyle diagram). E.g., users may want to be able to work alone or in a group, depending on what part of a task they are addressing. Additionally, workstyle models show how users might be constrained to work. For example, a mobile user may not have a network connection available, constraining him/her to work independently even if he/she preferred to work collaboratively. Similarly, a user working at home might prefer to use free-form drawing, but be constrained to using a structure editor due to lack of an electronic whiteboard.

In the case study of the Software Design Board, workstyle analysis helped us understand that to support software design, we needed to design tools supporting the brainstorming workstyle, and that we needed to reduce impedance of moving from hand-drawings to a CASE tool. The former goal was addressed by developing a whiteboard-based tool in which hand-drawings could be electronically captured and shared with other designers. The second goal was addressed by inclusion of a UML recognition tool and by the development of XML document interchange facilities allowing SDB designs to be exported to the Rosetta CASE tool. It is important to note that it is not reasonable to attempt to build a single tool supporting all workstyles. Rather, designers should attempt to support a range of workstyles, and provide low impedance mechanisms for transferring this work to other tools.

Other Techniques: The workstyle model complements a number other techniques for interactive system evaluation and design. For example, Lumsden has developed a sophisticated tool for matching CASE tools to their use (Lumsden and Gray, 2000). Design notations such as OPAS (Dubois et al., 1999) and our own dimension space (Graham et al., 2000) help in understanding the context of use of interactive systems, particularly how they fit within their physical environments. The Questions-Options-Criteria (QOC) method (McLean et al., 1991) helps evaluate design choices. While complementing these approaches, the workstyle model has the contribution of being simple to apply, and clearly showing where interactive system designs match or fail to match their intended work context.

6. Conclusions

In this paper, we have presented the *workstyle model,* a model capturing aspects of the style in which people carry out their tasks. The workstyle model complements task modeling by helping to develop deeper understanding of how tasks are to be performed. The model illustrates the ways in which people communicate during their work, and the properties of the artifacts that people produce.

We have shown that the workstyle model can be used to evaluate existing tools, and to help in the design of new tools. By examining the work styles supported by a tool, we can better understand how the tool can be effectively deployed, or identify mismatches between a tool and its intended use. By understanding the workstyles that a tool is to support, we can build tools that will be more effective in their deployment.

Acknowledgements

This work was partially supported by the Communications and Information Technology Ontario (CITO) Software Design Technology project and IBM's Centre for Advanced Studies. We gratefully acknowledge the donation by SMART Technologies Inc. of the SMART Board shown in figure 1. We thank Fonseca and Jorge for the use of the CALI gesture recognition system and Hopkins for the use of the Pie menu package.

References

Annett, J. and Duncan, K.D., (1967) Task analysis and training design. *Occupational Psychology,* 41:211-221, 1967

Begole, J., Struble, C.A., Shaffer, C.A. and Smith, R.B. (1997) Transparent Sharing of Java Applets: A Replicated Approach, In *Proceedings of the 1997 Symposium on User Interface Software and Technology (UIST'97),* ACM Press, NY, pp. 55-64.

Bly, S.A. (1988) A Use of Drawing Surfaces in Different Collaborative Settings. In Proceedings of the *Conference on Computer-Supported Cooperative Work (CSCW'98).*

Card, S. K., T. P. Moran, A. Newell. (1983). The psychology of human-computer interaction. Hillsdale, NJ, Lawrence Erlbaum Associates.

Chmura, L. and Norcio (1986) A. Design Activity in Developing Modules for Complex Software. In *Proceedings of Empirical Studies of Programmers,* pp. 99-116.

Damm, C.H., Hansen, K.M., Thomsen, M. (2000) Tool Support for Object-Oriented Cooperative Design: Gesture-Based Modelling on an Electronic Whiteboard, in *Proc. CHI 2000,* pp 518-525.

DeMarco, T., and Lister, T. (1987) *Peopleware.* Dorset House, New York.

Diaper, D. (1989) *Task analysis for human computer interaction,* Ellis Horwood.

Dubois, E., Nigay, L., Troccaz, J., Chavanon, O. and Carrat., L. (1999) Classification space for augmented surgery, an augmented reality case study, in *Proc. INTERACT '99.*

Finkelstein, A., Gabbay, D., Hunter, A., Kramer, J. and Nuseibeh, B. (1994) Inconsistency Handling In Multi-Perspective Specifications, *IEEE Transactions on Software Engineering,* 20(8), pp. 569-578.

Fonseca, M. J., Jorge, J. A., (2000). *CALI: A Software Library for Calligraphic Interfaces.* Tech. rep., DEI/IST/Technical University of Lisbon.

Graham, T.C.N., Stewart, H.D, Kopaee, A.R. and Ryman A.G. (1999) A World-Wide-Web Architecture for Collaborative Software Design. In *Proceedings of Software Technology and Engineering Practice (STEP'99),* IEEE Press.

Graham, T.C.N., Watts, L., Calvary, G., Coutaz, J., Dubois, E., Nigay, L. (2000) A Dimension Space for the Design of Interactive Systems within their Physical Environments, in *Proc. Designing Interactive Systems,* pp 406-416.

Hartson, H.R., Siochi, A.C. and Hix D. (1990). The UAN: A user-oriented representation for direct manipulation interface designs. *ACM Transactions on Information Systems,* 8(3):181-203

Hopkins, D. (1991) The Design and Implementation of Pie Menus. *Dr. Dobb's Journal,* December, Special issue on user interfaces (see www.piemenu.com)

Iivari, J. (1996) Why are CASE Tools Not Used? *Communications of the ACM,* October.

Jarzabek, S. and Huang, R. (1998) The Case for User-Centered CASE Tools. *Communications of the ACM.*

Johnson, P., Johnson H., Waddington, R. and Shouls, A. (1988). Task related knowledge structures: Analysis, modeling and application. In D.M. Jones & R. Winder, editors, *People and Computers: From research to implementation.* Cambridge: Cambridge University Press, pp. 35-62

Jones, T.C. (1986) *Programming Productivity.* McGraw-Hill, New York

Kraut, R. E. and Streeter, L. (1995) Coordination in Software Development, *Communications of the ACM,* 38(3), pp. 69-81

Landay, J.A. and Myers, B.A. (1995) Interactive sketching for the early stages of user interface design. In *Proceedings of CHI '95: Human Factors in Computing Systems,* Denver, CO, May, pp. 43-50.

E. Lank, J. Thorley, S. Chen, D. Blostein (2001) On-line Recognition of UML Diagrams, In *Proc. Sixth Intl. Conf. on Document Analysis and Recognition (ICDAR 2001),* Seattle, Washington, IEEE Computer Society Press, pp. 356-360.

Lumsden, J., Gray, P. (2000) SUIT - Context Sensitive Evaluation of User Interface Development Tools, in *Proc. DSV-IS'2000,* Springer LNCS, pp.79-95.

Malone, T. W. and Crowston, K. (1990) What is coordination theory and how can it help design cooperative work systems?, in *Proceedings of the Conference on Computer-Supported Cooperative Work,* pp. 357-370

McLean, A., Young, R.M., Bellotti, V.M.E. and Moran, T.P. (1991) Questions, options and criteria: Elements of design space analysis. *Human-Computer Interaction,* 6, pp. 201-250.

Paternò, F., Mancini, C., Meniconi, S. (1997) ConcurTaskTree: a diagrammatic notation for specifying Task Models. In *Proceedings of Interact'97,* pp.362-369, Chapman and Hall.

Paternò, F. and Santoro, C. (2000) Integrating Model Checking and HCI Tools to Support Designers in Verification of User Interfaces Properties, In *Proceedings of DSV-IS'2000,* Springer LNCS, pp. 135-150.

Pederson, E.R, McCall, K., Moran, T.P., Halasz, F.G. (1993) Tivoli: An Electronic Whiteboard for Informal Workgroup Meetings, INTERCHI'93, pp 391-398.

Quatrani, T. (1998) Visual Modeling With Rational Rose and UML, Addison-Wesley.

Rumbaugh, J., Booch, G., Jakobsen, I. (1999) *The Unified Modeling Language Reference Manual.* Addison Wesley.

Seaman, C.B. and Basili, V.R. (1997) Communication and Organization in Software Development: An Empirical Study. *IBM Systems Journal* 36(4).

Tang, J.C. (1991) Findings from Observational Studies of Collaborative Work. *International Journal of Man-Machine Studies*.

Tang, J.C and Minneman, S. (1991) VideoWhiteboard: video shadows to support remote collaboration, in *Proc CHI'91,* pp.391-398.

Vessey, I. and Sravandapudi, A.P. (1995) CASE Tools as Collaborative Support Technologies. *Communications of the ACM,* 38(1), pp., 83-95

Wharton, C., Rieman, J., Lewis, C., and Polson, P. (1994) The Cognitive Walkthrough Method: A Practitioner's Guide. In *Usability Inspection Methods,* J. Nielsen and R.L. Mack (Eds.), New York: John Wiley & Sons, pp.105-141.

Wu, J., Graham, T.C.N, Everitt, K., Blostein, D. and Lank, E. (2002) Modeling Style of Work as an Aid to the Design and Evaluation of Interactive Systems. To appear in the proceedings of CADUI'02.