

Tool Support for Cooperative Object-Oriented Design: Gesture Based Modeling on an Electronic Whiteboard

Christian Heide Damm, Klaus Marius Hansen, Michael Thomsen

Department of Computer Science, University of Aarhus

Aabogade 34, 8200 Aarhus N, Denmark

{damm, marius, miksen}@daimi.au.dk

ABSTRACT

Modeling is important in object-oriented software development. Although a number of Computer Aided Software Engineering (CASE) tools are available, and even though some are technically advanced, few developers use them. This paper describes our attempt to examine the requirements needed to provide tool support for the development process, and describes and evaluates a tool, Knight, which has been developed based on these requirements. The tool is based on a direct, whiteboard-like interaction achieved using gesture input on a large electronic whiteboard. So far the evaluations have been successful and the tool shows the potential of greatly enhancing current support for object-oriented modeling.

KEYWORDS

Gesture input, electronic whiteboards, cooperative design, object-oriented modeling, user study, CASE tools.

INTRODUCTION

Software developers use models to develop object-oriented software. In the early stages of a software development project, developers focus on understanding the part of the world that the computer system should support. Throughout the project, they represent their understanding in the form of models. The models are not only used in order to understand and discuss the world, but are also implemented in code and thus form an important part of the final software system.

A variety of Computer Aided Software Engineering (CASE) tools have been created to support the developers' work throughout the development process [12]. However, in practice these tools are supplemented with whiteboards, especially in creative phases of development.

The most appealing aspects of whiteboards are their ease of use and their flexibility. Whiteboards require no special skills, they do not hamper the creativity of the user, and they can be used for a variety of tasks. Their many

advantages aside, for most development projects whiteboards are not enough. Capturing diagrams electronically with CASE tools facilitates code generation, documentation, and allows developers much more flexibility in editing and changing the diagrams. The conflicting advantages and disadvantages of whiteboards and CASE tools can lead to frustrating and time consuming switches between the two technologies. Our goal is to design a tool that offers the best of both worlds.

Paper Structure

The next section presents the motivation for our design. We then discuss two user studies from which we derive a set of design implications. We describe the Knight tool we developed based on these observations, and present an evaluation of the tool. Finally, we discuss directions for future research and draw our conclusions.

BACKGROUND

Use of whiteboards has been studied in various contexts including meeting rooms [9][15], classrooms [1], and personal offices [17]. Whiteboards are very simple to use and many activities are ideally suited for this simple interaction. Computational augmentation [24] can potentially solve problems with whiteboards such as lack of space and efficient editing facilities. We are concerned with the use of whiteboards in a specific work practice, cooperative object-oriented modeling, and the potential use of augmentation in that setting.

CASE tools seek to support software development techniques such as diagramming, code generation, and documentation. Nevertheless, adoption of CASE tools in organizations is slow and partial [5][8]. A main reason is that current CASE tools are targeted at technically-oriented methods rather than user-oriented processes. In particular, CASE tools are weak in supporting creativity, idea generation, and problem solving [7].

The *Tivoli* system [20][16] inspired us and was a starting point for our work. It is designed to support small, informal meetings on an electronic whiteboard. The similarity of user interaction to that on ordinary whiteboards is stressed. In order to be able to support specific meeting practices, *Tivoli* introduced domain objects that allow customizations of the tool to support, e.g., brainstorming sessions and decision-making meetings. We focus on the creation and

manipulation of a certain kind of domain objects and the integration of these into a computational environment.

OBSERVING DESIGN IN PRACTICE

We conducted two field studies of software developers using CASE tools and whiteboards in order to understand the current practice of object-oriented modeling. In both studies, we observed a group of developers with mixed competencies and then interviewed them. The developers used the Unified Modeling Language (UML [22]), which is a formal graphical notation containing several different diagram types. We concentrate on UML class diagrams, which are used to model central concepts and relationships found in the real world (or an imagined world).

Each study focused on three aspects of the design activity: *cooperation*, *action*, and *use*. *Cooperation* includes the communicative, coordinative, and collaborative aspects of design. *Action* and *use* are akin to the categories Bly et al. used in their observations of shared drawings [3]. *Action* involves the physical interaction of the designers with tools. *Use* involves the semantics of the result of actions.

User Study 1: Building a New System

Research setting. COT [29] is a technology transfer project between Danish universities and private companies. As part of COT, a university research group and developers from a private company cooperatively designed an object-oriented control system for a flow meter.

Participants. The developers from the private company had no previous knowledge of object-oriented development, whereas the university research group consisted of experienced object-oriented developers. The developers from the private company acted as domain experts in initial phases of development, while the university researchers were object-oriented software designers and, to some extent, mentors for the developers from the private company. During the two-week period in which the project was studied, the number of people attending design meetings varied. Typical sessions involved 2-4 developers from the private company and 2-4 university people.

Procedure. The sessions took place in meeting rooms equipped with multiple ordinary whiteboards, an overhead projector, and a computer. We observed three sessions in detail. In each of these an observer took notes.

Observation 1: Alternating Between Tools

Tom, Mike and Peter¹ are about to discuss a new area of the problem domain. To brainstorm an initial design, Tom and Mike draw initial models on a whiteboard. Just before lunch they run out of whiteboard space and Peter captures the work so far using a digital camera. After lunch Tom and Mike continue on a freshly-wiped whiteboard, while

Peter redraws the diagrams from before lunch in a CASE tool using the photos as a reference.

Developers alternated between whiteboards and CASE tools. A typical work sequence involved sketching a model and then transferring it to a CASE tool, which could then use the formal model to generate code.

The next morning, Mike uses the CASE tool to generate diagrams from existing code. These illustrate details he and Tom discussed the previous day. Mike places printouts of the diagrams on the overhead projector (Figure 1) and Tom uses whiteboard markers to make amendments.

Work on an area of the problem domain involved several cycles of drawing and redrawing both on whiteboards and in CASE tools.

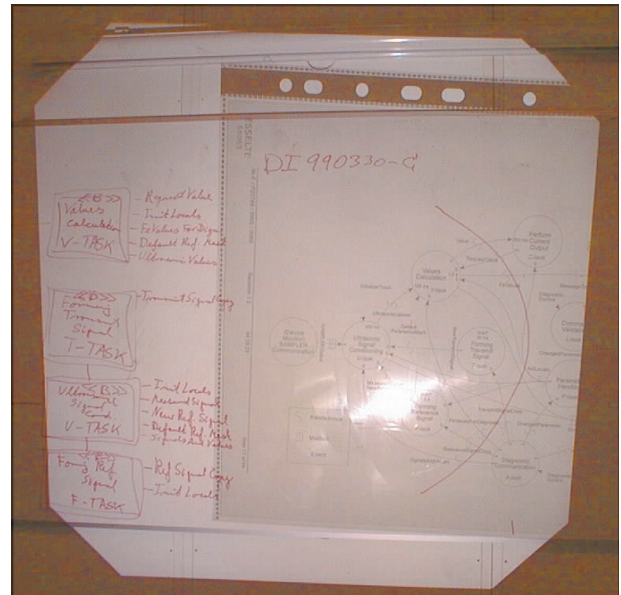


Figure 1. Using transparencies on a whiteboard

Observation 2: Working with a Formal Notation

Tom explains how a flow meter works conceptually. As he explains, he tries to model this using a UML diagram containing classes and relationships. He stops several times in order to ask Mike and Peter how to draw UML elements correctly. Moreover, to understand the semantics of the drawings, Mike and Peter often interrupt Tom.

The formal UML notation was hard to learn for the inexperienced developers. These syntactic problems caused a number of interruptions and breakdowns during modeling.

Peter and Tom are modeling how a number of objects interact with each other in the system. They want to show how these are synchronized. However, the UML is incapable of describing these aspects. Mike suggests a new notational element for this. Peter and Tom pick this up and use it in subsequent design problems.

The semantics of the notation was extended in order to make it support the work process and to add expressive

¹ The names of participating designers have been changed throughout this paper.

The developers frequently cooperated by taking turns at the blackboard. Figure 3 shows two developers standing at the blackboard, taking turns adding, deleting or changing the model. Abrupt interruptions were rare and only one developer drew on the model at any time. However, other discussion often took place while another person was drawing.

Design Implications

The two user studies highlighted the effectiveness of ordinary whiteboards as tools for cooperative design. They support a direct interaction that is easy to understand, and they never force the developer to focus on the interaction itself. Whiteboards allow several developers to work simultaneously and thus facilitate cooperation. They do not require a special notation and thus support both formal and informal drawings. Notational conventions can easily be changed and extended.

Whiteboards, however, miss several desirable features of CASE tools. Without the computational power of CASE tools, making changes to the drawings is laborious, the fixed space provided by the board is too limited, and there is no distinction between formal and informal elements. There is also no support for saving and loading drawings.

These observations led to the following design criteria for a tool to support object-oriented modeling:

- *Provide a direct and fluid interaction.* A low threshold of initial use is needed and the tool should never force the developer to focus on the interaction itself. The whiteboard style of interaction is ideally suited for this.
- *Support cooperative work.* Several developers must be able to work with the tool cooperatively. Informal cooperative work with domain experts as well as software developers must be supported.
- *Integrate formal, informal, and incomplete elements.* Besides support for formal UML elements, there must be support for incomplete UML elements and informal freehand elements. Also, the support for formal UML elements must be extensible, to allow for the introduction of new formal elements.
- *Integrate with development environment.* Integration with traditional CASE and other tools is needed. Diagrams must be saved and restored, and code must be generated and reverse engineered.
- *Support large models.* A large workspace is needed. In addition, there must be support for filtering out information that is not needed at a given time.

DESIGN OF THE KNIGHT TOOL

Based on the user studies, we have designed and implemented the *Knight* tool. The Knight tool uses an electronic whiteboard, currently a SMART Board [26], as its input medium. The SMART Board is a 72-inch touch-sensitive computer screen mounted in a cabinet to resemble a

traditional whiteboard. Users can draw on the surface using a number of pens (or just using their fingers). In contrast to other electronic whiteboards, such as, e.g., the Xerox Liveboard [20], the SMART Board unfortunately only allows input from one pen at a time.

The prototype is implemented in Tcl/Tk [19] with the [incr Tcl] extension [14], runs on the Windows and Unix platforms and is available for download from the Knight homepage [25]. We kept the interface very simple (Figure 4). A large workspace, resembling an ordinary whiteboard, is the central part of the user interface. The interaction is based on pen-strokes made directly on the workspace.

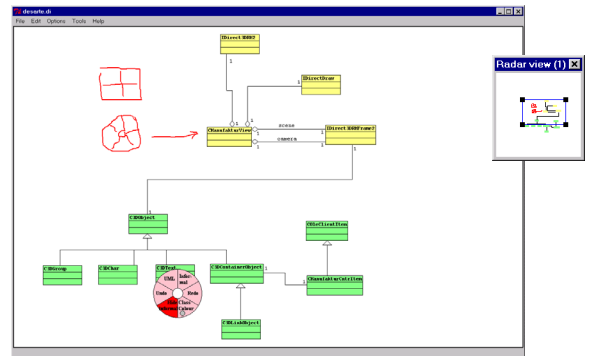


Figure 4. Knight user interface

Formality, Informality, and Directness

We wanted the tool to support a continuum of drawing formality, ranging from informal sketching elements over incomplete UML elements to formal UML elements. To allow this, the tool currently operates in one of two modes: Freehand mode or UML mode. We recognize that the use of modes is potentially problematic. However, our studies indicate that users already naturally operate in these two different modes, in different phases of the design. Freehand mode supports idea generation and UML mode supports design formulation. Two different background colors indicate the different modes.

In freehand mode, the pen strokes are not interpreted. Instead, they are simply transferred directly to the drawing surface. This allows the users to make arbitrary sketches and annotations just as on an ordinary whiteboard. Unlike on whiteboards, these can be moved around or hidden. Each freehand session creates a connected drawing element that can be manipulated as a single whole.

In UML mode, pen strokes are interpreted as gestural commands that create UML elements. If, e.g., a user draws a box, the tool will immediately interpret this as the gesture for a UML class and replace the pen stroke by a UML class symbol (Figure 5).

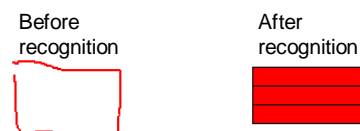


Figure 5. Recognition of the gesture for a UML class

The diagrams need not adhere to the UML semantics completely, in that incomplete diagram elements are allowed. Figure 6 shows how the user can input a relationship between two classes with only one of the two classes specified. The relationship can later be fully specified.

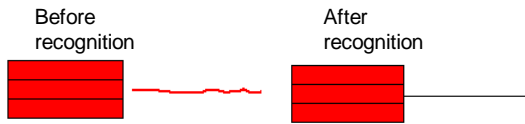


Figure 6. A relationship with only one class specified

The gestures for creating UML elements have been chosen so as to resemble what developers draw on ordinary whiteboards. This makes the gestures direct and easy to learn.

Another set of short directional gesture strokes chooses operations from a number of marking menus [10] illustrated in Figure 7.

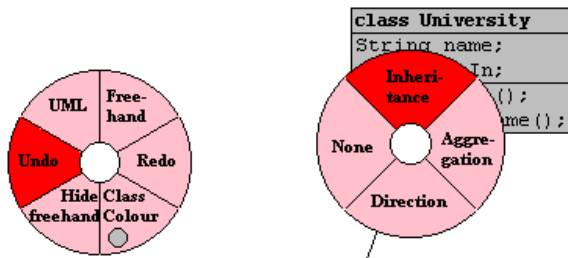


Figure 7. Context-dependent pie menus

For example, in order to undo or redo, the user may either make a short left or right stroke, or press and hold the pen and choose the corresponding field in a popup pie menu. The marking menus are also used to switch between UML and freehand mode. Marking menus support the interaction on a large surface well, because they are always ready at hand, unlike usual buttons and menus [20]. Apart from supporting a transition from initial to expert use, the marking menus also conveniently provide an alternative way of creating certain diagram elements (Figure 7 right). The marking menus are context-dependent. Depending on the immediate context in which a stroke or press was made, it will be determined whether it should be interpreted as a gesture command or as a marking menu shortcut. In the latter case a context-specific menu will be shown.

Use of Gestures

We use Rubine's algorithm [21] to recognize the gestures. This algorithm has the advantage of being relatively easy to train: To add a new gesture command, one simply draws a number of gesture examples. Potential problems with the algorithm, and gesture recognition in general, include that only a limited number of gestures can be recognized and that no feedback is given while gestures are drawn. To address these problems, we use *compound gestures* [11] and *eager recognition* [21], respectively.

Compound gestures combine gestures that are either close in time or space to a diagram element. For example, a user can change an association relationship (represented by an undecorated line) to a unidirectional association (represented by a line with an arrowhead) by drawing an arrowhead at the appropriate end. In this way, users can gradually build up a diagram, refining it step-by-step.

With eager recognition, the tool continuously tries to classify gestures as they are being drawn. Whenever the gesture can be classified with a high confidence, feedback is given to show that the gesture was recognized, and the rest of the gesture is used as parameters to the recognized gesture's command. For example, when a move gesture is recognized, the elements located at the starting point of the gesture will follow the pen while it is pressed down.

Support for Large Models

The workspace is potentially infinite, allowing users to draw very large models. It also supports zooming, as in zoomable interfaces [2]. In order to provide overview and context awareness, one or more floating "radar" windows can be opened (Figure 8; see also Figure 4 upper right).

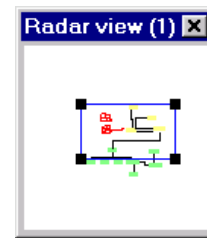


Figure 8. Radar windows provide context awareness

These radar windows show the whole drawing workspace, with a small rectangle indicating the part currently visible. Clicking and dragging the rectangle pans while dragging the handles of the rectangle zooms. By opening more radar windows, multiple users can have convenient access to pan and zoom, even though the physical drawing space is large.

Filtering is in a preliminary stage. Currently, it is possible to suppress details of the formal UML model and toggle the visibility of informal elements.

Tool Integration

The Knight tool must be integrated with existing CASE tools, to facilitate code generation from the models. Although the Knight tool is currently only able to exchange data with the WithClass CASE tool [27], we are currently working on making it a plug-in front-end to a variety of tools. In this way it is possible to use the CASE tool capabilities to create or edit models outside a cooperative modeling situation.

EVALUATION OF THE KNIGHT TOOL

We evaluated the current design of the Knight tool in two sessions. Both sessions were actual design sessions in which Knight was the primary tool. The purpose was to evaluate the usability of the tool in a realistic work setting.

First, a facilitator introduced the Knight tool to the participating designers and taught them the basic use of the tool. During the evaluation, he also helped if the designers had problems and asked for help.

We videotaped the sessions and took notes. As in our user studies, we focused on three aspects of design: cooperation, action, and use. Following the design sessions, we conducted qualitative interviews.

Both sessions were encouraging. Each lasted approximately one and a half hours, and the participants were able to maintain focus on their job, rather than on the tool or the evaluation.

Next we discuss the results of the evaluations with respect to the design criteria identified and summarized in “Design Implications” above.

Evaluation 1: Designing a New System Using Knight

Research setting. The CPN2000 project [6][31] is concerned with developing and experimenting with new interaction techniques for a Petri Net editor with a complex graphical user interface. The original user interface is a traditional window-icon-menu-pointer interface, whereas the new interface will use interaction styles such as tool-glasses, marking menus, and gestures. As part of the design, three object-oriented models for the handling of events and for the implementation of certain interface elements had been constructed. We observed the meeting in which these three models were integrated into one model using the Knight tool.

Participants. Three designers participated in the meeting. One of these had modeled the event handling and was knowledgeable of UML and traditional CASE tools. The other two modeled the interaction styles and had little knowledge of UML.

Results

The resulting diagram is shown in Figure 9. This rather large model was constructed with few problems and mishaps.

Provision for a direct and fluid interaction. After the short introduction, the participants were able to use the tool for long periods without help: the tool had a low threshold for initial use.

The use of gestures was mostly unproblematic. However, some participants had trouble drawing certain gestures. This may be due in part to too little training, but the gesture set can also be improved. For example, people draw differently with respect to size, orientation, and speed, and the gesture examples used to train the recognizer must encompass such variations.

Support for cooperative work. The electronic whiteboard worked well as a center of cooperation. The only problem participants reported was that only one person could draw

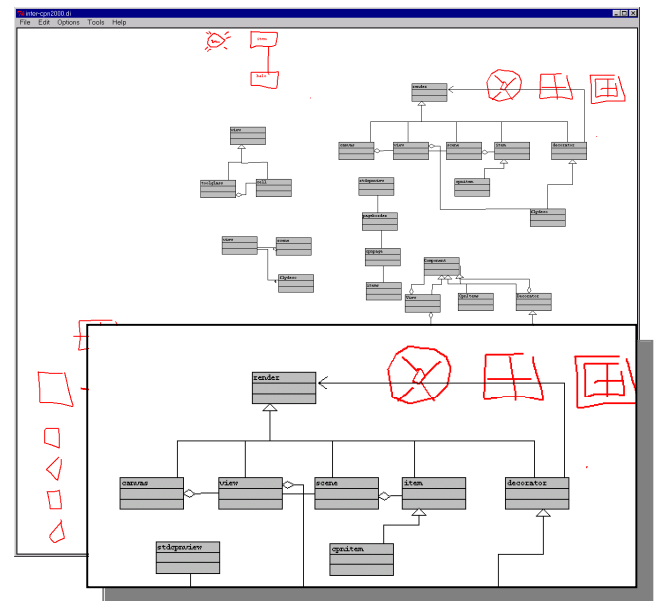


Figure 9. Diagram produced in the first session (with a blow-up of the upper right part)

at a time. Nevertheless, each developer was able to hold his or her own pen, and they all coordinated their actions when necessary.

Integration of formal, informal and incomplete elements. Freehand drawings were widely used and appreciated. The low resolution of the actual electronic whiteboard meant that the freehand drawings were relatively coarse-grained. In addition, response from the electronic whiteboards was delayed when a user drew quickly. This meant that freehand text was hard to do both legibly and quickly.

Support for large models. As the size of the model grew, the radar window was used to pan and zoom efficiently. However, the fact that the radar window was the only way to move around the workspace was problematic. In order to move a class from one corner of the large diagram to another, one participant had to move, pan, and then move again. This suggests the need for some other means of scrolling.

Evaluation 2: Restructuring a System Using Knight

Research setting. The DESARTE project [32] is concerned with designing an electronic support environment for architects. As part of this environment, a 3D replacement of the workstation desktop is implemented. A conceptual model had previously been designed and was to be restructured during this meeting using the Knight tool.

Participants. Two designers attended the meeting: The designer responsible for implementing the 3D desktop and a user involvement expert with an understanding of architectural work practice. Both had a good knowledge of object-oriented modeling.

Results

The second session showed a few more breakdowns and problems than evaluation 1. The developers were nevertheless able to complete the session and their work.

Provision for a direct and fluid interaction. After a short time, the participants were able to use the tool without many problems. When an error did occur, such as the system interpreting a gesture differently than expected, the participants sometimes got confused about what was happening: The feedback of the tool was not sufficient in the event of misinterpretations.

One of the participants initially had many problems operating the marking menu: Often he invoked commands by accident when drawing. This was partly due the fact that he had no previous knowledge of gestural input and marking menus.

Support for cooperative work. The two participants had no trouble cooperating around the tool.

Integration of formal, informal, and incomplete elements. The participants often made freehand drawings to illustrate the user interface of the designed environment. A minor problem in this case, was that that informal and formal elements could only be rudimentary connected, and there was little support for advanced grouping. Incomplete UML elements were considered useful, but were not widely used.

Support for large models. In this evaluation, the focus was on restructuring an existing diagram of moderate size, and the radar window was mostly used for zooming.

DESIGN IMPLICATIONS & FUTURE WORK

The observations and subsequent interviews showed that the Knight tool is a valuable tool for modeling in practice. However, as the above results point out, improvements are needed.

A number of physical problems with the actual electronic whiteboard hindered cooperation. Only one person at a time could draw on the whiteboard and informal drawing was only slowly rendered. The latter problem may be handled in part by the Knight tool, whereas the former is intrinsic to the specific electronic whiteboard. However, the lack of support for synchronous drawings was not construed as a major problem in the two evaluations. Since design processes are becoming increasingly distributed, we are currently investigating distributed cooperative design using the Knight tool. Integration with a mediaspace [4] may provide a non-intrusive way of supporting distributed communication in relation to this.

Problems with gestures caused a number of breakdowns. Several possibilities exist for alleviating this. First, more appropriate feedback can be given when a gesture has been drawn. Second, personalized gestures may be necessary, e.g., in the form of a *personal pen*, as in Tivoli [20]. For

each personal pen there could be a separate gesture set, a separate mode, separate colors, or other personal settings.

The integration of formal, informal, and incomplete elements is not complete. It is not, e.g., possible to connect formal and informal elements. A dynamic extension of the formal notation is a step towards this integration. The environment, with gesture recognition based on examples and an interpreted programming environment, makes such extensions technically feasible. *Flatland* [18] defines non-overlapping *segments* with different behaviors. Such segments may be used to group formal and informal elements separately. A notion of overlapping groups may be used to link these different segments.

Many of our observations of object-oriented modeling seem to be true for other types of formal modeling such as task modeling. A natural step would be to implement support for these as well, especially if combined with the idea of overlapping groups. This would facilitate combination of informal elements with formal elements, as well as handling types of formal elements together.

Filtering should also be considered in depth. Especially in evaluation 1, after the diagram had reached a certain size, navigation in the workspace became time-consuming. It should be possible to selectively hide parts of a model and give drawing elements temporality so elements may exist only for a certain period of time.

An important future area of research is the use of the Knight tool as a plug-in interface for different tools, which we are currently working on. This will involve longitudinal studies of the use of the Knight tool in development projects.

CONCLUSION

We have developed a tool for object-oriented development: Knight. The design of the tool is based on user studies of software developers creating object-oriented models. These show that important design criteria for a usable tool are (1) a direct and fluid interaction, (2) support for collaborative work, (3) an integration of both formal and informal drawing elements, (4) support for modeling in the large and (5) integration with existing development tools.

The Knight tool was designed to meet these criteria by using a large electronic whiteboard as input medium and by using an interaction style similar to that of traditional whiteboards. Input is done using gestures that resemble what is drawn on whiteboards. Both formal and more informal elements are supported and several developers can easily cooperate at the electronic whiteboard. Knight thus maintains the advantages of whiteboards and additionally adds features only possible in a computer based tool: Models can be easily modified, diagrams can be exported and imported to and from CASE tools, elements can be hidden and later restored, and a much larger workspace is provided.

ACKNOWLEDGEMENTS

We thank Michael Tyrsted who participates in the Knight project. We also thank Wendy Mackay for many discussions and for critique and help in writing this paper. Furthermore, we thank Michel Beaudouin-Lafon as well as the people from Danfoss Instruments, Mjølner Informatics, the DESARTE project, and the CPN/2000 project.

The Knight Project is carried out in the Centre for Object Technology that has been partially funded by the Danish National Centre for IT Research [28].

REFERENCES

1. Abowd, G., Atkeson, C., Feinstein, A., Hmelo, C., Kooper, R., Long, S., Sawhney, N., Tani, M.: Teaching and Learning as Multimedia: The Classroom 2000 Project. *Proceedings of Multimedia'96*, 1996, 187-198.
2. Bederson, B.B., Hollan, J.D. Pad++: A Zooming Graphical Interface for Exploring Alternate Interface Physics. *Proceedings of UIST*, 1994, 17-26.
3. Bly, S.A., Minneman, S.L. Commune: A Shared Drawing Surface. *Proceedings of the Conference on Office Information Systems*, 1990, 184-192.
4. Bly, S.A., Harrison, S.R., Irwin, S. Mediaspaces: Bringing people together in a video, audio and computing environment. *Communications of the ACM*, 36(1), January 1993.
5. Iivari, J. Why Are CASE Tools Not Used? In *Communications of the ACM*, 39 (10), 1996.
6. Janecek, P., Rutzer, A.V., Mackay, W.E. Redesigning Design/CPN: Integrating Interaction and Petri Nets in Use. *Proceedings of the Second Workshop on Practical Use of Coloured Petri Nets and Design/CPN*, 1990, 119-133.
7. Jarzabek, S., and Huang, R. The Case for User-Centered CASE Tools. *Communications of the ACM*, 41 (8), 1998.
8. Kemerer, C.F. Now the learning curve affects CASE tool adoption. In *IEEE Software*, 9 (3), 1992.
9. Kraut, R., Fish, R., Root, R., Chalfonte, B. Informal Communication in Organizations: Form, Function and Technology. *Groupware and Computer-Supported Cooperative Work*. 1993, 287-314.
10. Kurtenbach, G. *The Design and Evaluation of Marking Menus*. Ph.D. Thesis, University of Toronto, 1993.
11. Landay, J.A., and Myers, B.A. Interactive Sketching for the Early Stages of User Interface Design. *Proceedings of CHI'95*, 45-50.
12. Lyytinen, K., Tahvanainen, V.-P. *Next Generation CASE Tools*. IOS Press, 1992.
13. Madsen, O.L., Møller-Pedersen, B., Nygaard, K. *Object-Oriented Programming in the BETA Programming Language*. ACM Press, Addison Wesley, 1993.
14. McLennan, M.J. [incr Tcl]: Object-Oriented Programming. In *Proceedings of the Tcl/Tk Workshop*, University of California at Berkeley, June 10-11, 1993.
15. Moran, T.P., Chiu, P., Harrison, S., Kurtenbach, G., Minneman, S., van Melle, W. Evolutionary Engagement in an Ongoing Collaborative Work Process: A Case Study. *Proceedings of CSCW'96*, 150-159.
16. Moran, T.P., van Melle, W., and Chiu, P. Tailorable Domain Objects as Meeting Tools for an Electronic Whiteboard. *Proceedings of CSCW'98*, 295-304.
17. Mynatt, E.D. The Writing on the Wall. *Proceedings of INTERACT'99*, 1999, 196-204.
18. Mynatt, E.D., Igarashi, T., Edwards, W.K., and LaMarca, A. Flatland: New Dimensions in Office Whiteboards. *Proceedings of CHI'99*, 346-353.
19. Ousterhout, J.K. *Tcl and the Tk Toolkit*. Addison-Wesley, 1994.
20. Pedersen, E.R., McCall, K., Moran, T.P., and Halasz, F.G. Tivoli: An Electronic Whiteboard for Informal Workgroup Meetings. *INTERCHI'93*, 391-398.
21. Rubine, D. Specifying gestures by example. *Proceedings of SIGGRAPH'91*, 329-337.
22. Rumbaugh, J., Jacobson, I., Booch, G. *The Unified Modeling Language Reference Manual*. Addison-Wesley, 1999.
23. Russell, F. The case for CASE. *Software Engineering: A European Perspective*. Thayer, R., McGettrick, A. (Eds.) IEEE Computer Society Press, 1993, 531-547.
24. Wellner, P., Mackay, W., Gold, R.: Guest Editors' Introduction to the Special Issue on Computer-Augmented Environments: Back to the Real World. In *Communications of the ACM*, 36(7), 1993.

ONLINE REFERENCES

25. <http://www.daimi.au.dk/~knight>
26. <http://www.smarttech.com>
27. <http://www.microgold.com>
28. <http://www.cit.dk>
29. <http://www.cit.dk/COT>
30. <http://www.mjolner.com>
31. <http://www.daimi.au.dk/CPnets/CPN2000>
32. <http://desarte.tuwien.ac.at/>