iBistro: A Learning Environment for Knowledge Construction in Distributed Software Engineering Courses

Andreas Braun
Accenture
Maximilianstr. 35
80539 München, Germany
andreas.braun@accenture.com

Allen H. Dutoit, Andreas G. Harrer, and Bernd Brügge
Technische Universität München
Institut für Informatik/ I1, Boltzmannstraße 3
85748 Garching b. München, Germany
{dutoit, harrer, bruegge}@cs.tum.edu

Abstract

We have taught several distributed software engineering project courses with students and real clients [4]. During these projects, students in Pittsburgh and Munich, Germany collaborated on the development of a single system. Our experiences showed that software development is communication intensive and requires the collaboration of many stakeholders. Communication is challenging in distributed contexts: participants do not all know each other and work at different times and locations; the number of participants and their organization change during the project; participants belong to different communities. Hence, to deal with the global market place, it is critical to provide students with distributed collaboration skills. To improve the teaching of collaboration in software engineering, we propose iBistro [2], an augmented, distributed, and ubiquitous communication space. iBistro aims to overcome problems resulting from miscommunications and information loss in informal or casual meetings. iBistro enables distributed groups to collaborate and cooperate in software projects and therefore provides an environment for learning in such diverse aspects as project management, programming skills, and social skills. With the addition of techniques from artificial intelligence, such as student modeling, and intelligent support mechanisms, such as computer supported group formation, distributed tutoring becomes feasible.

1. Teaching Informal Communication

The development of engineering products and software is becoming increasingly distributed. Participants located at different geographical sites have to collaborate to specify, design, realize, and test products, usually across several time zones and often without meeting each other in person. There are many reasons for the distribution of product development:

- As a result of mergers and acquisitions, corporate knowledge, person power, and skills are distributed along historical organizational structures and are rarely centralized in one location.
- As a result of new technology or new product lines, participants from different divisions and departments are required to collaborate on the same project.
- Part of the project must often be collocated with the deployment site to optimize communication with the end user.

Distributed projects leverage off tools, such as groupware, distributed repositories, and videoconferencing utilities, to accumulate and distribute knowledge and artifacts. Distributed projects, however, introduce many technical and social barriers. In addition to being geographically distributed, participants come from different corporate cultures, use different tools, follow conflicting standards, and often speak different languages. Such challenges are difficult to meet and often cause the failure of the project. Our goal in teaching software engineering at Technische Universität München (TUM) and at Carnegie Mellon University (CMU) has been to provide a realistic software engineering experience to students. We have done this by immersing students in a single, team-based, system design project to build and deliver a complex software system for a real client. Since Fall 1997, we had the opportunity to teach four distributed software engineering project courses in TUM and CMU [4]. Teams of students at CMU and TUM were taught to collaborate using groupware (e.g., web sites, Lotus Notes, Email) and configuration management systems (e.g., CVS) to design and build a system for a client. Client reviews and internal reviews were conducted using videoconferencing facilities, enabling each site to present its progress

and obtain feedback from the client and from the other site. While all four projects were completed successfully and students acquired a number of skills for dealing with distribution, we experienced many difficulties in the areas of communication and collaboration among the sites. In particular, participants at both sites spent much more effort during solving unexpected problems and interface mismatches than would have been the case in a single site setting. Distribution made the three following obstacles especially difficult:

- *Inability to find stakeholders quickly.* Since participants were distributed and did not know each other, finding the author of a piece of code or of a subsystem could take several days. Similarly, finding a project participant who had an area of expertise to help with a specific problem could likewise take several days.

- *Inability to access rationale knowledge.* Since many decisions taken by teams were taken during meetings or informal conversations, participants at the other site could not easily access the rationale of the system. Hence, participants encountered unexpected problems when enhancing or modifying components produced by the other site. While meetings were documented in meeting minutes that were available via the groupware, such records were organized chronologically and were difficult to search when looking for a specific problem.

- *Inability to identify the latest version of the software (or of individual components) quickly.* Even though participants used the same repository for tracking versions of their components, it was difficult to identify when new versions were checked in and which problems new versions addressed. Similarly, a site was usually not aware of whether a new version was under test and about to be released. Consequently, sites worked often on outdated versions and produced version conflicts by solving problems twice.

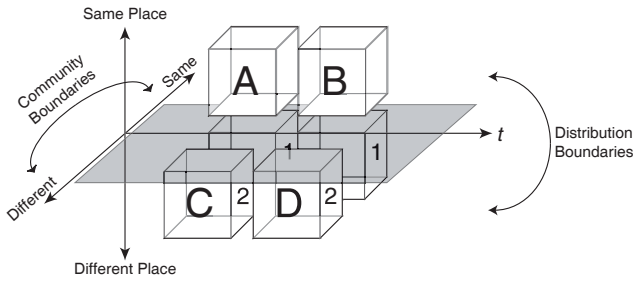
Note that all three problems noted above were caused, at least in part, by some type of communication breakdown. Researchers distinguish between informal and formal communication and recognize their application to different types of issues [12]. Formal communication is typically non-interactive and impersonal and includes, for example, formal specifications, written documentation, structured meetings. Informal communication is typically peer-oriented and interactive and includes, for example, hallway conversations, lunch breaks, and informal conversations that follow formal meetings. While formal communication is useful for coordinating routine work, informal communication is needed in the face of uncertainty and unexpected problems. Note that all three problems noted above were caused, at least in part, by lack of informal communication, typical of distributed projects [1, 5, 8, 12]. To address these problems in the classroom, we propose to further apply our approach of 'learning by doing'. In this paper, we describe iBistro, an experimentation environment that allows instructors and students to capture, structure, and retrieve knowledge produced during informal conversations.

iBistro will be used in the distributed project courses we run to encourage informal (single site or distributed) meetings and improve their record. By using iBistro, students will be directly exposed to distributed issues and made aware of several options for addressing them. Instructors will be able to improve their support and guidance for conducting informal meetings. By incrementally developing iBistro, we believe that instructors and students will collaboratively learn to deal increasingly more effectively with the issues of teaching communication and collaboration skills. This paper is structured as follows. Section 2 provides an overview of iBistro and an example scenario. Section 3 describes in more detail how iBistro can be used to capture, structure, and retrieve knowledge, and more generally, address problems such as the rationale retrieval and the version identification we identified above. Section 4 describes the integration of intelligent support in iBistro for addressing the stakeholder identification problem mentioned above. Section 5 lists our results achieved so far and Section 6 concludes this paper by outlining our experimental plans and the outlook for iBistro.

2. iBistro - An Augmented and Informal Meeting Space

iBistro is an augmented meeting space for informal collaboration and communication in distributed software engineering. iBistro therefore is a source for acquiring informal knowledge. The capturing and structuring of knowledge is facilitated by the automatic recording of many different types of related context. Examples of such context stored in iBistro include the identity of stakeholders/meeting participants, the participant's current activity, the current meeting topic, time, and location (e.g., the room where the meeting takes place). iBistro aims at supporting informal meetings in a distributed environment between two continents, just over the street, or even within the same room (see Figure 1). In addition, iBistro supports asynchronous collaboration by enabling participants to improve the content and the structure of meeting minutes. iBistro takes advantage of camera observed whiteboards and links to project documents to provide a seamless transition between the computer and the physical world. The coffee room provides these resources in an informal atmosphere while attempting to capture critical team interactions, allowing us to document some of the

rationale behind and the context surrounding a group decision to create a "group memory". Individual distributed developers (as opposed to groups of developers) are integrated from their workstation or wearable digital assistant with iBistro's ubiquitous interface for accessing the contextual memory over a network.



A.	Same time/ same place informal meetings.
B.	Different time/ same place meetings in the same (1) or different (2) communities.
C.	Same time/ different place meeting both in the same (1) and in different (2) communities.
D.	Different time/ different place meetings.

Figure 1. Distribution over space, time, and communities.

In this paper, we describe the features of iBistro for same place informal meetings, whether they occur synchronously or not and in the same community or not (Cells A, B_1, and B_2 in Figure 1). The same place/ different time meeting scenario describes the resumption of a meeting using iBistro's services. In this case, iBistro would provide participants with the agenda, stakeholders, topics, argumentation, rationale, or decisions of a preceded meeting. Once we understand same place meetings sufficiently well, we will refine iBistro's features to support distributed meetings (Cells C_1, C_2, and D in Figure 1).

The simplest form of a meeting within iBistro is the same time/ same place meeting (see A in Figure 1). Assume three meeting participants who are developing scenarios during a requirements elicitation session: Alice is a student who works in system development for this lab. Bob is the supervisor of the software development course and has some further domain specific knowledge. Claire represents the customer of the project. Using the electronic badges (such as the Active Badge Location System [18]) given to each of the participants at the beginning of the course iBistro knows about the presence of the individual stakeholders and is able to deal with a varying number of meeting participants. This is important to allow for the assignment of the content (e.g., a single requirement) to the individual stakeholders during the post-meeting process. Moreover, the system must not know about the participants in advance. This also fosters the informal character of the meeting. In our scenario, Claire is late for the meeting and arrives in iBistro after Alice and Bob have already started. Knowledge acquisition is done by capturing the audio and video of the meeting as well as by capturing any sketches, notes, and drawings made throughout the meeting. By allowing for the manual orientation of the whiteboard camera, meeting participants may use a whiteboard (including electronic whiteboards), a laptop computer, paper, or even napkins to draw or to write on. For this scenario, suppose that our meeting community uses an electronic whiteboard, such as SMART Boards^{TM}[16]. As the whiteboards' content is captured in a movie in iBistro (as opposed to whiteboard-capture systems such as Zombieboard [14], which use static images), the history of drawings or notes is saved in a sequence of images. Back to our scenario, Alice, Bob, and Claire are able to talk easily about the requirements of their project. However, they can also be sure that critical team interaction is saved and available for later processing and structuring. During the course of the meeting, two different application concepts are developed. The first one is a context sensitive computer advisor who guides the user by offering suggestions based on the state of their accounts with the bank. The second one, is a computer catalog that enables users to browse and search through the complete range of products offered by the bank. At a critical point during the meeting, Claire, the client decides to set aside the catalog concept in favor of the advisor concept. Alice decides to take a snapshot of the current state of both mockups by pointing the camera to a piece of paper on which they drawed some sketches. The whiteboard is then erased and the remainder of the meeting is dedicated to the advisor concept. After the meeting, during the post-processing, Alice navigates through the meeting record along its timeline using the MINUTEGENERATOR tool. However, the time needed for post-processing the meeting is shortened drastically by offering any contextual event as well as subsequent changing content of the whiteboard as an index into the captured meeting[1]. From Claire's late arrival to the meeting, Alice can easily distinguish between the strategy topics first discussed with Claire and the domain specific topics discussed after Claire's arrival. Also, by noticing the event associated with wiping the whiteboard, Alice is able to isolate the discussion associated with the two application concepts. She creates two option events representing each a concept and a decision event that she associates with the video segment when Claire made her decision. During this process, Alice creates the knowledge base for that meeting by evaluating the captured videos, therefore translating the lower- level captured information (such as audio and video streams and whiteboard snapshots) into higher level content (such as re-

[1] The acceptance of the time needed for post-meeting processing largely depends on the user group (consultants vs. students).

quirements and their rationale). One crucial point during that phase is that every information interpreted that way is linked automatically with its originating source and related contextual information, e.g., a stakeholder (identity) to allow for the later sorting of knowledge by different criteria (e.g., author, time of occurrence, type of event).

3. Knowledge Capture, Structure, and Retrieval in iBistro

3.1. The iBistro Architecture

The iBistro architecture is based on the *blackboard model* [15]. The blackboard model is originally used in opportunistic problem-solving to deal with non-computable and diverse problems in AI. We propose a blackboard-based architecture, called a *distributed concurrent blackboard model* as an approach to deal with the variety of events and context that occur during (informal) meetings [3]. In our architecture, various *capture components* record contextual events (such as people entering or leaving the meeting room) as well as audio, video, and whiteboard content. Several specialized *knowledge sources* seize the captured events to create new types of information or knowledge, for instance a hypothesis or a solution. This process of knowledge construction creates an abstraction hierarchy of knowledge (see Figure 3) that is stored in several *layers of abstraction* within the blackboard. While the *control* component notifies knowledge sources and schedules knowledge source invokation, the *strategy* component takes care of the "big picture" to direct knowledge acquisition. Figure 2 displays the simplified architecture for iBistro.

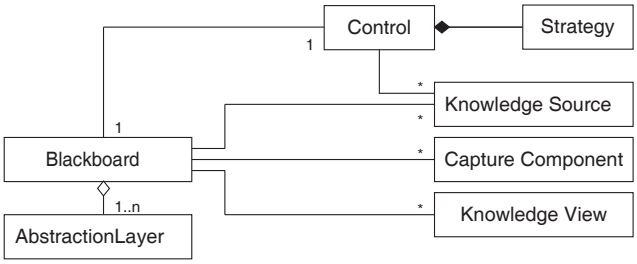


Figure 2. Simplified blackboard-based architecture for iBistro (UML class diagram).

3.2. Knowledge Capture and Storage

Knowledge acquisition and storage in iBistro is strictly event-based. Any type of knowledge in iBistro is stored according to its timely occurrence. Thus, the flow of events in a single meeting follows a common timeline. Incidents captured later, such as the manual post-interpretation of the whiteboard video, which might result in a (single) requirement, are added with a timestamp representing their post-meeting creation. Surrounding contextual information is linked to the event to indicate the originator (the identity of the person who mentioned the requirement), the time when the requirement was first mentioned, location, and so forth is linked along with the event in the database. The ability of context and event interlinkage aims at reflecting that "an activity happens within a context" [6] and vice versa.

Versioning of knowledge is handled similar to the linking of knowledge with related context. Each individual event might have several versions – successors and predecessors, e.g., different wording for a requirement. Each version, again, might have its own surrounding context, such as creation time, author, and location. This is relevant since work in iBistro can be distributed between several locations. This also implies links between distributed iBistros and will result in several distribution issues. Figure 3 depicts the event hierarchy used by iBistro together with its versioning mechanism.

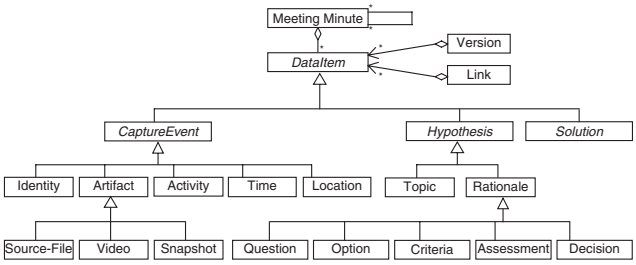


Figure 3. Knowledge database and content linkage for a single meeting (UML class diagram).

3.3. Knowledge Acquisition

Due to the result-oriented character of meetings in iBistro, the post-mortem structuring of captured knowledge is crucial. In general, this takes place after the client and user left a brainstorming session during requirements elicitation. The post-meeting structuring is typically performed by a person, called "meeting editor", (e.g., a consultant or developer). During the post-meeting, the editor annotates the captured audio and video stream with higher level information to provide an index into the raw material. In iBistro, we use the Question, Option, Criteria paradigm (QOC, [13]) as a basis for these annotations. The editor identifies topics that were discussed by attaching a Question event to a segment of the video. Within that segment, the editor identifies different alternatives with different Option events. The

editor documents a decision, such as the selection of an alternative or the discarding of others, by creating a Decision event. Since History events, like all other iBistro events, have an attribute identifying the originator of the event (in this case the person who suggested the option or who made the decision), traceability to human sources is ensured. In addition to History events, the editor can also attach other types of information using Link events, such as references to other material, for example, a problem statement from the client, scenarios, and questions generated by REQuest[2], or a class diagram generated by a CASE tool. Note that the editor does not need to view the entire record of the meeting to create useful meeting minutes. Moreover, other meeting participants could assume the role of editor afterwards and refine the minutes iteratively. The meeting editor idea originated from the experiences with knowledge management at Accenture, where a knowledge champion is responsible for saving relevant domain or project knowledge to Accenture's internal knowledge databases, called Knowledge Xchange™ (KX). KX is Accenture's worldwide knowledge management system. It became the backbone of the firm's global knowledge capital. Launched in 1992, the system today is the largest Lotus Notes installation worldwide with more than 65,000 users in 2000. However, iBistro brings this idea to a different level of granularity (knowledge capture per meeting vs. per project/ sub-project) and focuses on the kind of meetings typically omitted in KX. As these informal meetings are normally (intentionally) quite unstructured, the importance of the structuring process and a suitable tool-support is evident. Figure 4 shows a screen-shot of the MINUTEGENERATOR tool.

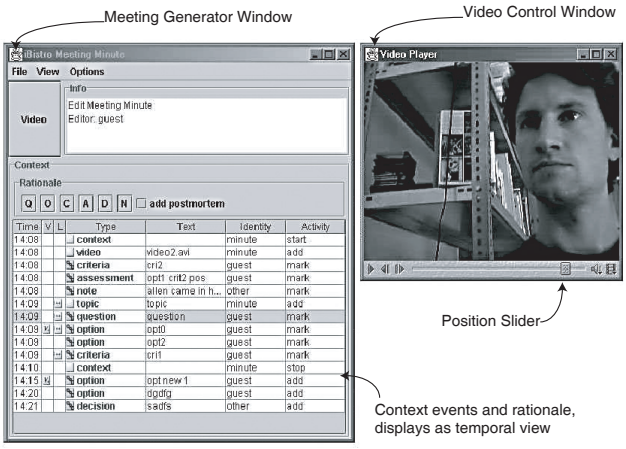


Figure 4. Meeting generator tool for post-meeting structuring.

[2]REQuest [7] is a Web-based tool for rationale-based use case specification. REQuest enables users propose requirements and their justifications, to review and to discuss them using the QOC paradigm.

After the post-mortem processs, the hierarchy of information and knowledge now stored in the meeting minutes can be translated to a three-dimensional model as shown in Figure 5. The three axes represent the timeline (x-axis), level of abstraction (represented in the blackboard layer, y-axis), and version or knowledge-interlinkage (z-axis).

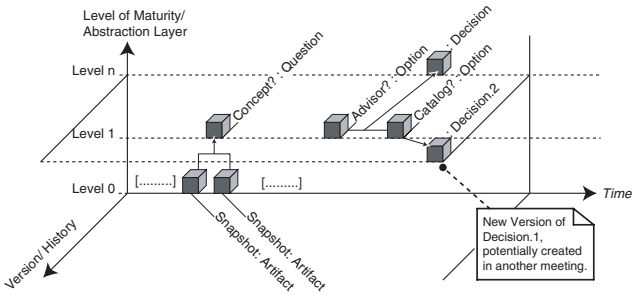


Figure 5. Information in iBistro seen as a 3D-model of knowledge.

3.4. Knowledge Retrieval

The meeting minutes stored in the repository represent the natural flow of the meeting, including external annotations from other sites or an individual's personal computer. A self-evident way to view such a meeting is to playback the meeting as a multimedia archive, thus enabling non-participants to access the raw information. In iBistro, the SMIL-MEETINGVIEWER generates on-demand a SMIL[3] [17] file (or data stream) to represent the meeting along with the captured requirements, context, rationale, and so on. This allows interested people to navigate through a meeting using any SMIL compliant video player, such as RealPlayer™ or Quicktime's Movie Player to view the meeting. As the content of the meeting follows a common timeline, the 'Clip Position slider is used to navigate through the captured audio, video, as well as other content such as requirements. Alternatively, the History events can be used to jump to specific segments of the meeting minutes, for example, navigating an option will move the position slider to the frame where the option was first suggested. Graphical views of requirements or rationale can be displayed using HTML or by generating bitmaps on demand. However, displaying multi-dimensional components, such as context-links between stored entries which allow navigation, is non-trivial. Thus, the next-generation meeting view (which actually doesn't replace the SMIL view, but will live side-by-side, solving different needs) will facilitate the

[3]SMIL™ (pronounced "smile") enables simple authoring of multimedia presentations over the Web. A SMIL presentation can be composed of streaming audio, streaming video, images, text or any other media type.

n-dimensional navigation through the captured knowledge from various sites. As currently implemented in the MINUTEGENERATOR, the knowledge base must be searchable by any type of context, e.g., by stakeholders, location, topic, versions, and so on. Hence, in addition to the raw context information captured during the meeting, the user also sees all the annotations and structure that were added during the post-processing.

3.5. Meeting Minute Navigation

As knowledge in iBistro is stored along with its related contextual information, navigation is possible using various types of input. As shown in Figure 3, meeting minutes consist of contextual information (e.g., location, identity, activity, history, and time) which can serve as keys for searching. For example, a minute may be sorted by requirements authored by a certain participant, by time, or any other key. Navigation is possible on any of those keys: the stakeholder of an issue is found by clicking on that issue. Related information, like time or location where the meeting took place, is displayed accordingly and might be used for further navigation. Thus, iBistro's database can be used to find stakeholders over various meetings or even projects. While a MEETINGVIEW provides a meeting-based index into the knowledge base, other knowledge sources can provide an artifact-based view into the knowledge base. For example, we plan to modify the REQuest requirements engineering tool so that developers can also browse meeting segments based on a specific scenario or use case. By providing a seamless integration between meetings, models, and documents, the value of iBistro will be more visible to the meeting participants.

4. Domain Expert Knowledge and Intelligent Support for iBistro

With techniques from intelligent tutoring systems (ITS) and computer supported collaborative learning (CSCL) systems we can enhance the learning environment with many facilities for support of working and learning processes: the means for this encompassing support are the use of student models, the explicit representation of expert knowledge in the problem domain (in our case software engineering), and the analysis of event traces to create higher-level information. Student or user models enable us to address one of the problems described in the introduction, that is the inability to find stakeholders quickly. With a combination of self-assessment from the user's side and diagnosis of the user's problem solving behavior we can get a representation of the user's capabilities, expertise and weaknesses, that is a user model or profile. This can be used to help the users finding stakeholders and experts in certain areas much more quickly, just by requesting help from the system to get a recommendation which person should be contacted. For example a student in the software-engineering course, whose task is the implementation of a subsystem, runs into problems with the design the team planned. At that moment he needs the help of a design expert in the team. Based on self-assessment of the students and on diagnosis of the previous work, iBistro could recommend a team member meeting the criteria the other student asks for. The process for finding specific stakeholders or roles is very similar to that in Opportunistic Group Formation [10], a well-known procedure in the field of computer supported collaborative learning. With the explicit representation of expert domain knowledge (here with the topic of software-engineering, like process models, rationale, design, and its refactoring) the learning environment iBistro may also provide intelligent support on its own, if a human expert is not available (due to asynchronous work or different time zones). The fundament of that expert knowledge is the definition of an ontology of the domain, which defines all the important terms and relations of the expert domain. At the moment we design an authoring tool for the definition of concept maps and ontologies. For the user-interface of the artificial domain expert we propose the technique of synthetic interviews and synthetic agents [11], that provide an artificial anthropomorphic partner for the human group members. In that way the iBistro learning environment can support the students directly as a learning partner, especially in the distributed time scenarios when human partners are not available. Another point where artificial intelligence (AI) techniques can augment the iBistro environment is the analysis of captured event traces: As presented in Section 3.2, iBistro creates a trace of the events happening within the enviroment. These rather low-level data can be processed and abstracted into higher level information, such as the degree of participation of specific members in a community or the diagnosis, that usually a member takes a specific role, like mediator. Such an analysis is also used in intelligent distributed learning environments [9] for the creation of high-level student model information. Such an automated analysis makes the post-processing much easier for the meeting editor, because he can rather focus on the essential aspects of the meeting than caring about bookkeeping of statistical information, such as participation of individual members, which can be automatically computed and processed by the environmnt. The process of minute generation is with that support less of a burden for the meeting editor. These additional uses of techniques from artificial intelligence, which have already been applied to intelligent tutoring systems and CSCW-systems, could provide a much more encompassing support than conventional groupware systems give.

5. Results and Current Status

iBistro is currently tested and improved in a small distributed software development project at TU München and the National University of Singapore. This setup revealed some technical difficulties and deficiencies, especially regarding audio and video quality due to limited bandwidth and camera and orientation problems. This shows the importance of local post-meeting processing, while communication then is based the electronic meeting minutes. The architectural model which has been developed for iBistro, however, prooved to be suitable for both experimentation with context-aware devices and a model-view-controller (MVC) based approach to knowledge retrieval and navigation.

The distributed setup also showed the strengths of iBistro compared to simpler electronic communication (such es email), as related material (source files, snapshots, ...) and surrounding information and knowledge (rationale, stakeholders, ...) are directly available to remote participants.

6. Conclusion

In this paper, we motivated the need to support informal communication in teaching software engineering. We also emphasized the importance of informal meetings in the context of enhancing an organizational memory. We propose to address some of the issues surrounding informal communication by supporting the efficient capture, structure, and navigation of meeting minutes and their integration into the long term project memory embedded in tools and documents. We described iBistro, a research and teaching environment for experimenting with technologies and techniques for achieving these goals. We will address or avoid obstacles encountered by other related efforts by 1. focusing on working (brainstorming) meetings, 2. by incrementally developing guidance and tool support with real users, and 3. by leveraging off context-based devices to enrich the meeting record and simplify the post-processing of meeting minutes.

References

[1] A. Al-Rawas and S. Easterbrook. Communication problems in requirements engineering: A field study. In *Proc. First Westminster Conf. Professional Awareness in Software Engineering*, Univ. Westminster, London, 1996.

[2] A. Braun, B. Bruegge, and A. H. Dutoit. Supporting informal requirements meetings. In *7th International Workshop on Requirements Engineering: Foundation for Software Quality. (REFSQ'2001)*, volume 7, Interlaken, Switzerland, June 2001.

[3] A. Braun, B. Bruegge, A. H. Dutoit, T. Reicher, and G. Klinker. Experimentation in context-aware applications. Submitted to HCI Journal for publication in special issue on context-aware computing, 2000.

[4] B. Bruegge, A. H. Dutoit, R. Kobylinski, and G. Teubner. Transatlantic project courses in a university environment. In *7th Asia-Pacific Software Engineering Conference*, Singapore, Dec. 2000. APSEC.

[5] B. Curtis, H. Krasner, and N. Iscoe. A field study of the software design process for large systems. In *Communications of the ACM*, volume 31(11), Nov. 1988.

[6] P. Dourish. What we talk about when we talk about context. Submitted to HCI Journal for publication in special issue on context-aware computing, 2001.

[7] A. H. Dutoit and B. Paech. Developing guidance and tool support for use case-based specification. In *Proceedings of the 7th International Workshop on Requirements Engineering: Foundation for Software Quality. (REFSQ'2001)*, volume 7, Interlaken, Switzerland, June 2001.

[8] R. Grinter, J. Herbsleb, and D. Perry. The geography of coordination: Dealing with distance in r& d work. In *Communications of the ACM*, 1999.

[9] A. Harrer. Analysis of social interaction for construction of group models. In *Proceedings of AI-ED 2001*, San Antonio, TX, USA, 2001.

[10] M. Ikeda, G. Shogo, and R. Mizoguchi. Opportunistic group formation. In *Proceedings of AI-ED 1997*, pages 167–174, Kobe, Japan, 1997.

[11] L. Johnson, editor. *Instructional Uses of Synthetic Agents*, LeMans, France, 1999.

[12] R. Kraut and L. Streeter. Coordination in software development. In *Communications of the ACM*, volume 38(3), Mar. 1995.

[13] A. MacLean, R. M. Young, V. M. Bellotti, and T. P. Moran. *Questions, Options, and Criteria: Elements of Design Space Analysis*, chapter 3, pages 53–106. Design Rationale: Concepts, Techniques, and Use. Lawrence Erlbaum Associates, Hillsdale, NJ, first edition, 1996.

[14] T. Moran, B. van Melle, and E. Saund. *Walls at Work – Physical and Electronic Walls in the Workplace*, pages 191–208. Deutsche Verlags Anstalt, Stuttgart, 1999.

[15] P. Nii. Blackboard systems: The blackboard model of problem solving and the evolution of blackboard architectures. In *AI Magazine*, volume 7(2), pages 38–53, 1986.

[16] SMART Board, 2000.

[17] W3C. SMIL. Technical report, World Wide Web Consortium, 1998.

[18] R. Want, A. Hopper, V. Falcao, and J. Gibbons. The active badge location system. In *ACM Transactions on Information Systems*, volume 10(1), pages 91–102, 1992.