CAIS: Collaborative Asynchronous Inspection of Software *

Vahid Mashayekhi
Chris Feulner
John Riedl

Department Of Computer Science
University of Minnesota
{vmash,feulner,riedl@cs.umn.edu}

Abstract

Many software engineering tasks have a synchronous component that requires the participants to assemble together at the same time and place. This approach is expensive in terms of traveling, scheduling and human resources. Existing computer tools mitigate these constraints by adding structure to the meeting, providing on-line document support, and distributing the participants over geographic boundaries. The constraint remains, however, that all participants participate at the same time

We propose relaxing the time constraint in software engineering tasks to resolve issues non-concurrently, in effect reducing (and in some cases eliminating) the need for the synchronous meeting. We hypothesize that support for asynchrony will enable software engineering teams to work together as effectively in different times as in same time.

We have chosen software inspection as our candidate software engineering task because it is well-understood, highly-structured, and widely-practiced. We have designed and developed a Collaborative Asynchronous Inspection of Software (CAIS) meeting prototype that supports the meeting part of inspection. CAIS allows participants to effectively "meet" even when separated by time zones and working schedules. We have conducted a pilot study comparing the manual and CAIS meetings and present our results and lessons learned.

Keywords: Concurrent Software Engineering, Asynchrony, Software Inspection, Computer-Supported Cooperative Work (CSCW), Collaboration, Notification.

1 Introduction

Humans solve hard problems by collaborating with one another. The traditional model of collaboration requires all participants to assemble together at the same time and place. This approach is expensive in terms of traveling, scheduling, human resources, pre-meeting preparation of material, and post-meeting recording of the results. Advances in distributed systems, networks, and user interface technology have helped *Computer Supported Cooperative Work* (CSCW) become a viable alternative to face-to-face meetings [4, 12, 37]. CSCW is the study of methods for enhancing cooperation among computer users by providing an infrastructure that explicitly supports the user interaction and sharing of information [20].

Software engineering is a domain in which support for collaboration can be fruitfully explored. Collaboration is a requirement in many software engineering tasks including requirements analysis, design, programming, debugging, and testing [18]. For most software engineering projects, analysts, designers, implementors, and testers must work together through an iterative process to build a software artifact. Research has shown that interaction among team members accounts for a significant part of the total cost of software systems [22]. It is our belief that more effective software engineering methods will impact the society at large through savings in time, money, and effort. Furthermore, we anticipate that solutions to the problems in software engineering will also apply to collaboration in other domains.

We have chosen software inspection as our representative software engineering task because it is widely-practiced and highly-structured. Software inspection is an effective method for detecting faults in documents and code produced in software development [25]. Boehm includes software inspection in his list of the ten most important issues for improving the quality of software, saying, "Walkthroughs catch 60 percent of the errors." [5].

Software inspection has evolved from a purely centralized and paper-driven process to one that can be

performed paperless over distance [31]. In distributed software inspection, participants can "meet" with people in other cities through workstations at their desks. The reduced travel costs increases the feasibility of inspection when the development team is not in the same location. On-line support aids the participants in document preparation and maintenance, eliminating unnecessary note-taking and duplication.

In this paper, we augment our previous work in distributing software inspection [31] to support asynchronous collaboration of inspection participants. Asynchronous, or non-concurrent, collaboration enables interaction without requiring all participants to be present at the same time. We find the concurrent participation in software inspection to be a significant component of the meeting cost and believe that asynchronizing the process can help in reducing the cost.

The time required from the meeting members and the problem of scheduling the meeting can make the synchronous meeting the bottleneck for the inspection. Meeting times not only depend on scheduling a conference room, but also on each member having the same block of time open in their schedules. Relaxing the time constraint allows the participants to work at a time of their choosing, providing a greater degree of freedom in their actions.
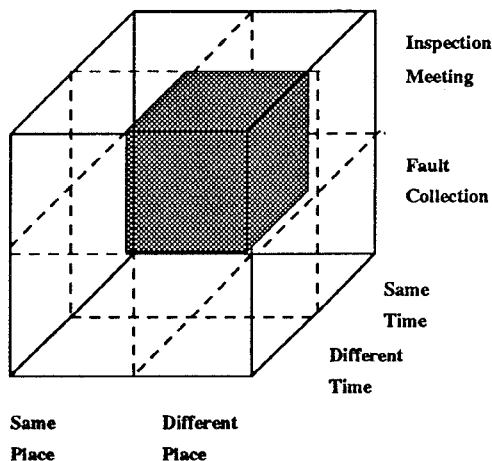

Figure 1: Time and Space Dimensions

In general, collaborative meetings can be categorized by the two dimensions of space and time [20]. A *same-time, same-place* meeting is the current board-room type meeting where everyone congregates at a table, using pointers, overhead projectors, and handouts as meeting tools. A *different-time, same-place* environment might be a bulletin board in a publicly accessible room used for posting announcements. An example of a *same-time, different-place* meeting would be a teleconferencing system supporting audio and video, allowing the meeting to take place with members in

different locations [35, 36]. Conceptually, a *different-time, different-place* meeting would enable participants to work together at the time and place of their choosing, but other than e-mail systems, existing implementations are scarce.

Ideally, to provide complete flexibility for the software development teams, support for dissimilar meeting modes must be available for the diverse set of constraints, requirements, and participation styles possessed by the different inspection teams. Their meeting tools must be able to supply as many of these meeting modes as possible.

The software inspection process consists of two distinct meeting modes: Fault collection and inspection meeting. These modes are denoted in Figure 1 by the bottom and top halves of the cube. During fault collection (bottom half), individuals review the documents independently and are not restricted by place and time [31]. Therefore, the fault collection activity spans the time and space axes and consists of all the parallel planes across the bottom half of the cube. The inspection meeting process (top half), consists of all participants discussing the correlated fault list generated by the reviewers. Traditional centralized inspection meetings cover the back left octant (same-time, same-place), while the distributed meeting covers the back right octant (same-time, different-place) [31]. We are exploring the potential for asynchrony in the meeting phase, denoted by the shaded section (different-time, different-place). Combining distributed software inspection with an asynchronous meeting would give complete flexibility in time and space for the software inspection process.

In addition to enhancing flexibility throughout for the participants, asynchronous meetings may alleviate some of the social problems that have been reported in synchronous meetings [14, 26]:

Free Riding: A subset of the group might not contribute to the task, relying on other members for contributing ideas.

Limited Air Time: Only one person can speak at a time, limiting the time each person can contribute.

Production Blocking: Individuals have to withhold their contributions until they get a chance to report them. During the holding time, they are not producing any new ideas and contributions, and may subsequently forget or decide not to offer their ideas.

Serial Thread of Execution: Since ideas are pursued serially, fewer kinds of ideas may be pursued.

Asynchronous meetings might reduce these problems by providing a structured context for the on-going meeting. A history can preserve the meeting data for either new participants to join midway, or for a new employee to review a past meeting in order to get up to speed on a project. Participants have an equal opportunity for airing their views, alleviating the production block-

ing suffered by limited air time problem experienced in same-time, same-place meetings.

As we study asynchrony in inspection meetings, it could be that eliminating face-to-face meetings is neither possible nor desirable. For instance, verbal intercourse is regarded to be critical to the task of software inspection as currently formulated [9], and this may be true for many types of meetings. Presently, our focus is on technical issues in distributed and asynchronous inspection. If our work is successful, it will make it possible for further investigation of the social changes introduced by an asynchronous meeting model. Our expectation is that face-to-face meetings may turn out to be critical for team-building and for resolving particularly different issues. However, we expect that, with suitable support, many day-to-day activities can be performed completely asynchronously.

We hypothesize that support for asynchrony will enable software engineering teams to perform most tasks together as effectively in different-time mode as in same-time mode.

We believe that asynchronous emulation of the inspection conversation structure is feasible using a structured history for discussing issues and putting forth resolutions, dynamic voting protocols for reaching consensus, and a notification sub-system for user coordination and awareness.

Section 2 introduces related work in collaborative systems. Section 3 presents the requirements and decisions made in the design of our prototype. Section 4 discusses possible implementations of CAIS in Electronic Mail, Mosaic, and Lotus Notes platforms and details our implementation in the Suite environment. Section 5 describes the methodology, quantitative and qualitative measurements, results, and lessons learned in our pilot study. Section 6 concludes our paper with a summary of the results and future work.

2 Related Work

This section focuses on application-level groupware systems. We present five classes of groupware systems: Electronic mail and bulletin boards, collaborative annotators, Group Decision Support Systems (GDSS), Computer-Assisted Software Engineering tools, and software inspectors.

Electronic Mail and Bulletin Boards: Two widely used computer mechanisms for supporting asynchronous collaboration are electronic mail and bulletin boards. These systems allow users to send messages to groups of users. Structured use of the electronic messaging systems has been shown to be an effective collaboration tool [28, 30, 39].

Collaborative Annotators: The Design Journal [11] is a hypertext system designed to facilitate the cap-

ture of early design deliberations. It is implemented using a specific method called Issue Based Information Systems (IBIS). It uses a semi-structured issue-position-argument framework to provide a team of designers support for the capture and recording of design document discussion, creation, and commitments. The PREP Editor is a collaborative writing environment that is designed to provide commenting and annotation capabilities [10]. The system provides a columnar structure that provides columns of text for comments, author intent, and the actual text being reviewed.

Group Decision Support Systems: The Minnesota GDSS project aims at conducting theoretical and empirical research in Group Decision Support Systems [19]. The work introduces the Adaptive Structuration Theory (AST), which focuses on how technology structures are applied in interpersonal interaction and the specific nature of appropriation patterns [15]. A multi-user software environment named Software-Aided Meeting Management (SAMM) serves as a vehicle for experimentation. Its U-shaped conference table has a terminal and keyboard for each group member to enter ideas, comments, votes, or notes. A large screen at the front of the room provides a shared focus for the participants by displaying a summary of group activities.

Project Nick studies the theory of meetings and defines the meeting types and classes [12]. The focus is on small face-to-face meetings specializing in exploration activities such as brainstorming, defining design structure, analyzing issues, and problem resolution. Meeting-aids include an electronic blackboard, interconnected PCs, and recording apparatus. The Colab experimental meeting room developed at Xerox PARC is designed for facilitating interaction in small working groups [37]. The room is equipped with workstations linked together over a LAN, a large touch-sensitive screen, and a stand-up keyboard. Cognoter is a Colab tool used to prepare presentations collectively. It provides support for brainstorming, organizing, and evaluating. The PlexCenter Planning and Decision Support Laboratory at the University of Arizona provides a large U-shaped conference table with 16 workstations depressed below the table for line-of-sight considerations [3]. A large-screen projection system displays screens of individual participants or a compilation of screens. The facility is used for electronic brainstorming and issue analysis among other group activities. A newer version of the facility called the Collaborative Management Room, designed to accommodate larger groups, introduces more recent technological advances [33].

Computer Assisted Software Engineering (CASE) tools: Asynchrony is supported in a variety of existing CASE tools. UNIX Systems such as Source Code Control System (SCCS) and Revision Control System (RCS) provide utility programs designed to

manage multiple revisions of source files, automate the storing, retrieval, logging, identification, and merging of versions, and maintain a history of previous versions [40]. Apollo's DSEE environment helps to archive previous versions of sources, control access to all versions, document the history of each file, build programs and individual components, rebuild programs with previously built components, build components concurrently on distributed nodes, and manage program releases [38].

Software Inspectors: Collaborative Software Inspection (CSI) [31] is a tool created to support distributed collaborative software inspection. CSI automates the inspection process, but its synchronous nature requires participation from all participants at the same time. ICICLE [9] is a system intended to support the complex set of tasks performed during code inspection. Like CSI, it assists individual users in the comment preparation phase of code inspection. The ICICLE meeting environment is synchronous also, with computer support aiding in making it a paperless meeting. Collaborative Inspection Agent [23] (CIA) uses ConversationBuilder [27] to develop a tool for synchronous inspection of all work products at various stages of the life cycle.

Electronic mail and bulletin boards show that asynchrony is a feasible abstraction for collaboration. Projects such as SAMM, Nick, Colab, IBIS, and Prep demonstrate that computer support facilitates group meetings if the participants are gathered at the same place and time. The collaborative annotators suggest that collaboration technology may be successfully applied to traditional domains such as writing and editing. CASE tools, such as RCS and DSEE, assist software engineers in working together asynchronously. The CSI and ICICLE Software inspection tools show that software engineering tasks, such as inspection, can benefit from computer support.

Our prototype, Collaborative Asynchronous Inspection of Software (CAIS) asynchronizes the meeting portion of software inspection. Our work extends electronic mail and bulletin boards by structuring messaging explicitly for inspection tasks. Similarly, our work differs from the work done in the GDSSs and collaborative annotators in that we are examining a specific, well-structured meeting application. We add to the existing plethora of CASE tools by introducing an inspection tool for participants distributed across time and space. We build upon the work done in software inspectors by introducing asynchrony as an abstraction for software inspection meeting.

3 Design of CAIS

We define a software inspection meeting to consist of a sequence of discussions. A discussion, in turn, is a sequence of comments, terminated by vote-taking. If the result of vote-taking is inconclusive, the sequence of comments is extended to accommodate more discussion of the fault. A discussion is closed when consensus is reached regarding that discussion. Within a discussion, all comments are ordered. Ordering may also exist at the discussions level, but preferably they can be held concurrently.

CAIS aims at automating the capture of the discussion comments, votes, and results so that the meeting can be recorded and held over a period of time without requiring concurrent participation of the meeting members. CAIS achieves this goal through the capture and display of a structured history of the meeting. Every meeting has an agenda, discussions and votes. The CAIS meeting structure calls for the participants to engage in an exchange of comments about a fault, terminating this exchange by a vote. Each member can record comments until one calls for a vote. If the vote result is inconclusive, the exchange of comments continues with a new sequence of comments. If the vote result is conclusive, that fault is considered to be resolved and the participants move to another unresolved fault.

3.1 Requirements

There are several requirements that need to be transferred from a same-time, same-place type meeting to different-time, different-place meeting. This sub-section identifies these requirements, with an eye toward their evolution and refinement as the result of lessons learned in our earlier user studies.

Causal Order: Activities in a synchronous software inspection meeting happen in a linear fashion, where each activity has a clear predecessor and a successor. For instance, a discussion on a fault is started by participants contributing ideas on ways of resolving the fault. These comments are made in a serial way, with the participants taking turns voicing their opinion. Upon reaching a consensus, the participants move on to the next fault, eventually discussing all faults in a sequential manner. Observing this sequence of events ensures that all participants share the same context for the meeting. When asynchronizing the software inspection meeting, the linear order of events during the discussion of a single fault needs to be kept, but the discussions may be held in parallel (though some discussions may have dependencies that requires their order to be preserved). The benefits of concurrent discussions include parallel threads of execution, and resolving the limited air time problem.

Structured History: An asynchronous meeting must keep a structured history for current participants and for later distribution and review. Our initial CAIS implementation maintained the history in a separate

24

file, mainly intended for metrics collection and review. A pilot test of that implementation revealed that users preferred to have the information about what was done, who had done it, and when it was done as part of the application state and not to have to consult a separate file to discover that information. In the present design of CAIS, we ensure that every relevant occurrence has a corresponding place in the structured history by keeping track of who, what, and when. This support enables a user to "visit" a meeting and follow the conversation as it has been happening, possibly over a period of days.

Train of Thought: We need to maintain the user's train of thought from one visit of the meeting to the next. Train of thought is sustained in a same-time, same-place meeting since only one discussion takes place at one time. In a different-time, different-place meeting, the participants typically visit each concurrent discussion to check for new comments made by others and to add their own. An earlier implementation of CAIS displayed all the meeting discussions (decided or undecided) as a large sequence within a single view, requiring the users to visually scan the file to locate a discussion of interest. In the present design, we have made the new information easily distinguishable from previously read information by displaying only the undecided discussions to the users, one discussion at a time.

Reaching a Consensus: Consensus is normally reached by putting forth a proposal and holding a vote. A mechanism must be provided to allow for the taking and counting of votes. In our earlier design, we required all the participants to vote on a fault before its status could be determined. Subsequent pilot studies revealed that this design decision led to situations where the meeting was tied up due to a slow participant. We have refined the design to allow for more flexible voting protocols, where the status of a fault can be determined by a majority of voters concurring on a resolution, even if some have not voted yet.

Visual Cues: Speakers often direct the attention of meeting participants through the use of pointers and visual cues in a same-time, same-place meeting, or What You See Is What I See windows in a same- time, different-place meeting, and this must be supported in a different-time, different-place meeting also. In a pilot study, our participants reported difficulties in correlating a fault to its place of occurrence in the document. The lack of visual cues required the participants to scan the target material visually to locate the fault, resulting in a distracting and time-consuming process. Our present design provides an automatic display and hilighting of a fault in the source document when it is being discussed.

Progress: In same-time, same-place meetings, participants can easily be asked to comment or vote, and are conscious of what is expected of them in way of
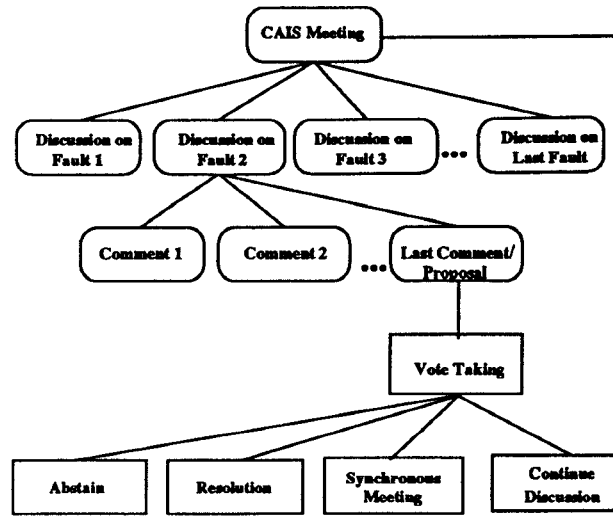

Figure 2: Conversation Structure in CAIS

contribution to the meeting. Our first CAIS prototype lacked these similar notions of deadlines or progress. Users had to visit CAIS to find out whether any new work was done since their last visit and had no idea about how much of the task was left to be done. Our current CAIS design provides participants with information on what has been done by others since their last visit, and what is needed from them to help contribute towards the conclusion of the meeting.

3.2 Conversation Structure

The CAIS meeting conversation structure is composed of three phases: Discussion of a fault until a potential resolution is identified, putting forth the resolution as a proposal, and quorum-based voting on the proposal (See Figure 2).

Annotations: CAIS borrows annotations from CSI [31]. During the fault collection phase, annotations are attached to the document being inspected. During the inspection phase, the annotations are organized into a set of discussions, with hyperlinks to lines in the original document. The annotations help satisfy the "Visual Cues" requirement.

Discussion: A CAIS meeting is consisted of a number of discussions. Each discussion pertains to a single fault and is comprised of any number of comments regarding that fault. The participants engage in a dialogue in an attempt to resolve the fault. Their dialogue is captured in a structured manner to allow each participant pursue the line of reasoning offered by the other participants. The CAIS discussions satisfy the "Structured History" and "Train of Thought" requirements.

Proposal: The exchange of comments during the discussion phase ends when a participant puts forth a proposal to the group. Further discussion of the fault

25

is disallowed until the status of the proposal is determined. Similar to the discussions, the text of the proposal and its status are saved as part of the history of the fault, satisfying the "Structured History" and "Train of Thought" requirements.

Vote: The participants vote on the proposal. The outcomes of their votes include accepting the proposal as a resolution for the fault, requesting continued discussion of the fault in the asynchronous meeting, abstaining from evaluating the proposal, and sending the fault to the synchronous meeting. Voting determines the status of the present proposal and satisfies the "Reaching the Consensus" requirement.

3.3 Notification Sub-system

In a same-time meeting, the participants use verbal communication to inform one another of their individual progress, and consequently are able to measure their collective progress towards the completion of their assigned task. Asynchronous collaboration closes this line of communication to its users. In a different-time meeting, computer support must provide the necessary awareness of the participants' activities and report on the state of the task.

We believe a notification sub-system to be a necessary component of any asynchronous collaborative environment. A notification sub-system can help by keeping the users aware of the most urgent matters, reducing the information volume, processing and presenting the information in a digestible form, and offering advice on how to coordinate participants' activities to better complete the task. The notification sub-system satisfies the "Progress" requirement by letting participants know who has done what, and providing them with information on what they need to do in order to complete their task.

The information that we report back to the user may be one of three types (See Figure 3):

Micro signifies small detailed changes in the state of information. Any unit of work, such as entering a comment or registering a vote, is considered to be micro information. Micro presents the information in its finest level of granularity and is appropriate for users who wish to be tightly-coupled with the other users. For instance, after a participant visits CAIS, all the other participants receive a micro notification message that textually details all the comments and votes recorded by the visiting participant in that visit.

Macro compiles a series of small changes in the state of information into a more digestible and understandable form. An example is a summary of user activities, such as the number of votes registered during a CAIS visit. Macro is a packaging of the micro information into a shortened, summarized form. It is appropriate
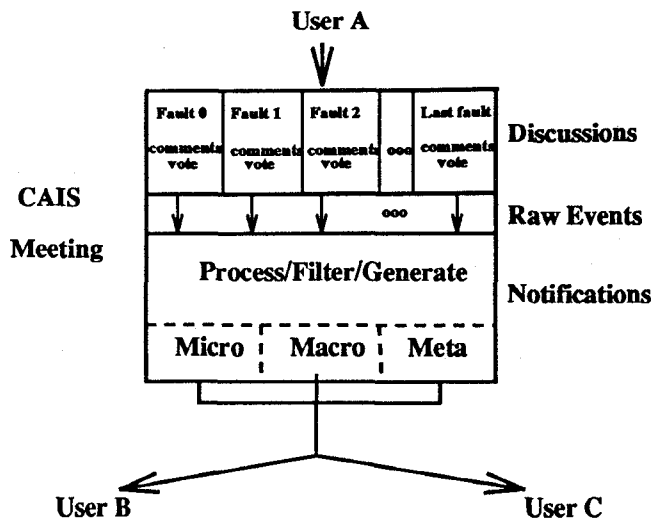

Figure 3: Notification Sub-System in CAIS

for users who are loosely-coupled with the other participants and do not wish to have intimate knowledge of every single action performed by them.

Meta provides information on the task, such as the completion percentage, or impending deadlines. Meta is an interpretation of the micro and macro information in order to estimate group progress and ensure timeliness in meeting deadlines. It may be used for arranging visitation schedules for the participants.

Our notification algorithm follows these steps: (1) Becomes aware of the completion of an event. (2) Determines whether the event necessitates notification of other users. This determination may depend on specific knowledge of the task, ratio of the completed work to total work, or the participants' availability schedules. (3) Determines user's course of action. For instance, one course of action may be preparation of a visitation schedule for the user to complete the task in a timely fashion. (4)s Send user the notification.

Our notification sub-system sends the three individual messages pertaining to micro, macro, and meta information. Users are notified based on either progress or visitation: Progress notifications are sent when so many units of work are completed by a participant, while visitation notifications are triggered every time a participant visits CAIS. If a participant requests further discussion of a fault that another participant has already voted on, the voter is notified of this request and of the cancellation of the vote. At the closure of a meeting, all participants are notified of the agenda of a possible synchronous meeting.

4 Implementation

There exist a wealth of software infrastructures for developing CAIS [1, 2, 8, 16, 21, 27, 29, 32]. In the fol-

lowing subsections, we describe how four selected platforms, namely Electronic Mail, Mosaic, Lotus Notes, and Suite, can be fruitfully explored for developing CAIS and detail our implementation in the Suite environment.

4.1 Electronic Mail

Arguably, electronic mail (e-mail) has been the most successful groupware system ever introduced [39]. Traditional e-mail systems are characterized by passive, uni-directional exchange of electronic messages between a sender and one or more receivers. Multimedia e-mail systems augment this exchange by supporting non-textual data, such as displaying of images and playing of audio, but the process has remained one-way and non-interactive [6, 7]. Computational or active e-mail systems propose to further facilitate asynchronous collaboration of users by embedding programs in electronic messages [8]. Active messages interact with the recipients of the messages and take different actions based on the recipients' responses. Though issues of portability, security, and standardization remain to be resolved, computational e-mail has several note-worthy capabilities that may be used for developing CAIS:

Distribution: E-mail systems support delivery of messages to remote users located at any place on the Internet.

User Interface: E-mail systems provide an interface to their users that supports composing, editing, deleting, browsing, saving, and archiving of electronic messages.

Computation: Computational e-mail systems offer facilities for performing actions on the user's behalf. For instance, consider a computational message that arrives at the receiver's end and engages the recipient in a question/answer dialogue to ascertain whether the user is available for a meeting. The responses from all the recipients may be sent back to the original sender, who is in charge of scheduling a meeting time for all the people involved.

Consider the following scenario of an inspection meeting using computational e-mail: The target material is sent to all participants, who review the material and send back their list of faults to the producer. The producer correlates and sends back this list to all the reviewers. The group then engages in an exchange of messages, discussing each fault. At the conclusion of each discussion of a fault, a vote template, embedded within a computational e-mail message, may be sent to the reviewers to capture their votes. The exchange of messages will continue until all the faults are addressed and their status is determined. Existing e-mail systems should be augmented in several ways to support CAIS meetings:

- Communication between participants must be structured according to software inspection guidelines.

- The shared information must be managed in a structured manner to aid the participants in following the discussions and the history of the meeting.

- Group decision making must be made available in e-mail.

Proposed active e-mail systems could be used to support these extensions for effective CAIS [8].

4.2 Mosaic

Mosaic is a networked information discovery, retrieval, and collaboration tool [2]. It is accessible across several platforms (X Window System, Microsoft Windows, and Apple Macintosh), capable of supporting multiple media, user-tracking, annotations, and document cross-linking. Mosaic provides a number of facilities that may be used for developing CAIS:

Distribution: Mosaic uses a client/server model of interaction. A server sits on a machine at an Internet site responding to queries sent by Mosaic clients from anywhere on the Internet.

Persistence: A unit of information, also called a document, may reside on local or remote file system. Using a Uniform Resource Locator (URL), a document anywhere on the network can be located. URLs can be thought of as a networked extension to the standard filename concept. They can describe anonymous ftp-able files, gopher documents, news groups on UseNet, files in a directory on a local or remote machine, or documents written in the HyperText Markup Language.

User Interface: A graphical user interface supporting font and style selection, cut-and-paste editing operations, user navigation, and history tracking is provided. A document, which can be of a variety of types including text (plain, rich, or hypertext), audio, video, image, or a graph, can be displayed graphically in the Mosaic browser.

Group Annotations: A group of users, distributed across the network, can collaboratively annotate a document. (Although this support is missing from the current version of Mosaic and is only available as part of an earlier version of the Mosaic browser and server software.)

Consider the following scenario of an inspection meeting in Mosaic: The target material, converted into HTML format, is available anywhere on the network and can be loaded into a Mosaic client. The user reviews the document and attaches group annotations to the document. Note that the granularity of an annotation is the entire document, and not lines or words of

27

the document. Other reviewers can read the group annotations by the first reviewer and add their own group annotations regarding the document. The annotations are time-stamped and bear the name of the annotator to help in building a context around the group work. There are a number of additional capabilities that would make Mosaic more suitable to support CAIS:

- Annotations need to be of a finer granularity and be attached to a line or word of the document. They also need to be anchored to the point of discussion in the document. Currently, they are either placed at the bottom or top of the document). This structuring allows users to read all the relevant annotations about a fault before contributing their own ideas. Fill-out forms are an extension of the Mosaic browser that support user input and could be used to support annotations.

- Evaluation of proposals through vote-taking needs to be added.

- Users need to be notified of new annotations, and votes made to the document.

4.3 Lotus Notes

Lotus Notes [29] manages information for a group of people distributed across a computer network. This management includes the ability to collect, organize, share, process, and customize information. Notes provides most of the capabilities required for implementing CAIS:

Distribution: Notes is a client/server model with the database containing the forms and documents for user access.

Persistence: Notes automatically provides protected and persistent objects.

Consistency: Upon commit of a comment or fault in Notes, the data is immediately available to other users on that server, and later, at a previously determined interval, on replicated servers. Asynchronous propagation of database updates (comments/faults) is an integral part of Notes.

Security: Access control lists and encryption provide security at many levels of granularity in Notes. Many different "roles", such as Depositor, Reader, Author, Editor, Designer, and Manager are available for assignment.

User Interface: Forms provide the user interface through which users view and edit the underlying data.

Consider the following scenario of an inspection meeting in Notes: For fault collection, a form is created with one visible field containing a single line of code of the document under review. A view is created that displays each one of these lines in sequence. A response document is created for the form, so that the reviewer could compose a document to enter a fault for that line of code. This document, when saved, can automatically collapse (not be visible in the view). During the discussion phase, a view is created which contains all of the unresolved faults. Another form allows the reviewers to enter comments under the faults. This form will have a "doclink" back to the line of code where the fault was recorded. When a fault is resolved, a field is checked to remove the fault and its descendants from the view. Notes mail in conjunction with periodic macros can handle the notification required in an asynchronous meeting. The moderator will have a view for resolved faults and those that need to be sent to the synchronous meeting. Notes can easily print views and forms for the paper documents needed for the meeting.

4.4 Suite

Suite is a software system for developing multiuser applications. A prototype of Suite has been implemented on top of UNIX, TCP/IP, NFS, and X [17]. The Suite object model is an extension of UNIX, allowing distributed, shared, protected, and persistent objects. The components of Suite are:

RPC: Suite RPC allows for applications executing in different address spaces and possibly on different hosts to name and communicate with each other by calling high-level remote procedures.

Persistence: Suite objects are persistent in that their data structures can be checkpointed onto disk and later restored to memory.

User Interface: Suite user interface treats all objects as data that can be edited. Interaction with "editable objects" is made possible by dialogue managers (DMs). DMs display a presentation of selected data structures, allow users to edit the presentation in a syntactically and semantically consistent fashion, and communicate these changes with the object. The object in turn ensures that the other displays also update their values.

4.4.1 CAIS Implementation

We have used the Suite software development environment [16] for developing CAIS. Suite was chosen as the application development platform foremost because of our success in building other multi-user applications in it, including CSI [31]. We use CSI for the individual reviews and fault collection. CSI supports collaboration of geographically distributed individuals in the inspection and provides on-line capability for recording and correlation of faults. The correlated faults are sent to the asynchronous meeting, where CAIS is used for the discussion and resolution of the collected faults.

4.4.2 Objects

CAIS consists of three objects that are briefly described next (See Figure 4):

Browser Object contains the document under review, with each line numbered. We support hi-lighting and automatic scrolling to a particular line of the document to aid the users in locating the line in question.

History Log Object records and time stamps the user activities for later analysis and review. The data collected include the total meeting time, participants' visitation schedules, time spent in discussions and votes, and number of comments and votes entered.

Meeting Object provides a hierarchy of faults, discussions, and comments. This structure is based loosely on a common meeting framework: A person introduces a fault, any number of people comment on it, a proposal is made to end the discussion, and a vote is taken. If the vote is conclusive, the discussion is ended and the next fault is introduced. Otherwise, the discussion on the fault continues. At the conclusion of a CAIS visit or if enough progress is made, notifications are sent to the participants using the mail delivery system. These notifications include a detailed listing of comments and votes entered by the current participant, a summary of all participants work in their most recent visit, and an evaluation of how much of the total task has been completed up to this point. The notifications make each user aware of individual and group progress.

4.4.3 Voting and Consensus

After a proposal has been suggested, the voting process begins. Our choice of a voting protocol in CAIS is only one possibility in a rich voting protocol space. We have based our decision on simplicity and repeatability of results. Each CAIS participant can vote in the following ways:

Agree indicates that the voter agrees with suggested proposal.

ContinueDiscussion indicates that the voter wishes to continue the discussion on the fault.

SendToSyncMeeting indicates that the voter believes that the fault cannot be resolved in the asynchronous meeting and should be sent to the synchronous meeting.

Abstain indicates that the voter wishes to be excluded from the voting process for this particular proposal.

In a same-time, same-place meeting, the votes can be counted publicly or secretly. In a different-time, different-place meeting, users are presented with a window in which they can enter their vote. In order to accept the proposal, the votes must be counted, and the results are based on the consensus method used; examples are unanimous, majority, or user-defined. A different-time, different-place meeting must be able to support any combination of variables, without forcing the participants to use one or the other and the vote must be persistent until all have voted, consensus is reached, or some other condition is satisfied. An additional problem that arises is that the slowest participant holds up an entire discussion. This is eased somewhat since all participants can attend the other discussions while one vote is pending. If the majority vote mechanism is used, the vote will end if a majority is reached, even if not everyone has voted.

4.4.4 Implementation Decisions

During a standard face-to-face meeting, the meeting protocol can change dynamically. Time limits may be imposed, new members may get to vote, or the voting method may change. This is also a desirable property for an asynchronous meeting. The owner or meeting manager should be allowed to dynamically change these options based on their own preferences. For CAIS, several of these properties have been fixed for the users in order to be able to take consistent measurements for the pilot study. These options are: (1) A majority vote in agreement is sufficient to end a discussion. We have decided on this form of reaching agreements dynamically to avoid having the meeting tied up by a slow participant. (2) One vote of **SendToSyncMeeting** ends the discussion and puts it on the agenda for the synchronous meeting. This is for hard problems that may be better handled face-to-face. (3) A vote of **ContinueDiscussion** opens the discussion of a fault again and informs the participants who have already registered votes that the discussion has been extended. This decision follows the current face-to-face meeting model where a discussion may continue as long as participants are interested in pursing it. (4) All faults not acted upon during the asynchronous meeting are sent to the synchronous meeting. This is also for hard problems for which time expires, and that may be better completed face-to-face.

5 Pilot Study

The primary goal of our pilot study is to assess the feasibility of distributing the software inspection meeting across time by comparing the manual inspection meetings with CAIS meetings. We include both quantitative measurements that include time measurements and comment counts, and qualitative measurements that include feelings about the meetings, and usability issues.

5.1 Methodology

Our pilot study involved Computer Science graduate students at the University of Minnesota. These stu-
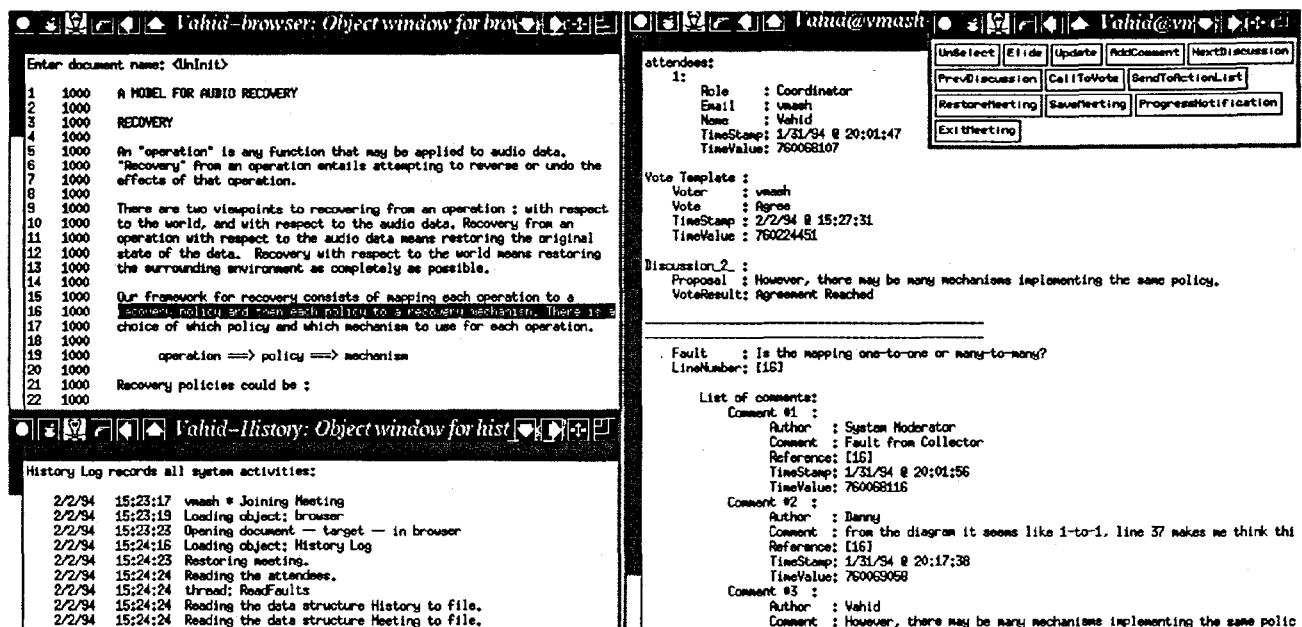
Figure 4: CAIS Windows

dents reviewed the requirements documents for two projects under development in our collaborative systems research group. We have chosen to inspect software requirements since requirements analysis is a pivotal stage in the software development waterfall model where many faults could be detected and corrected [13]. One group of students first manually inspected one requirements document, and then used CAIS to inspect the other, while the second group first used CAIS for inspection and then manually inspected the other requirements document. Each group acted as the control for the other group on one of the two target materials. The hardware used included Sun Sparc IPXs, Sun Sparc SLCs, and Sun Sparc 1s. Participants used machines located in the laboratories within the Computer Science building at the University of Minnesota or dialed in remotely from their workplaces or homes. Participants had different schedules and lifestyles: Part-time students working full-time as employees in research and financial industries, full-time students doing collaborative research, and full-time students in other varied research fields.

5.2 Quantitative Measurements

The CAIS History Log provides the following metrics: Number of faults discussed, number of comments per person, number of votes per person, time for individual fault collection, total meeting time, and visitation schedules.

5.3 Qualitative Measurements

We measure the qualitative characteristics of the prototype by asking the subjects to complete questionnaires after the manual and CAIS meetings. For both meeting types, we inquire about their degree of satisfaction with the meeting experience, level of agreement with the meeting structure, level of flexibility provided in their participation schedule, quality of discussions, ability to maintain their train of thought during the meeting, degree of participation in the discussions, percentage of time spent on non-task related issues, and participants' meeting preference.

5.4 Results

This sub-section presents the results obtained from the manual and CAIS meetings and questionnaires.

Number of Faults Discussed: Participants discussed all the faults noted during the review phase of the manual and computer-supported meetings. More faults were recorded, however, when participants used CAIS. Approximately, 15% of the faults discussed in the CAIS meetings were difficult to resolve asynchronously. These fault were set aside for synchronous meetings that were carried out at the conclusion of the CAIS meetings. Our analysis of the faults shows that all the faults sent to the synchronous meeting were categorized as "major" by the reviewers, indicating that it may be possible to filter out such faults prior to the CAIS meeting and send them to the synchronous meeting directly.

Number of Comments and Votes: The number of comments and votes recorded were comparable between

the two meetings, with the producers having a larger share of the total.

Time for Individual Fault Collection: On the average, the participants spent less than an hour in fault collection for both meeting types. This time was spread over a two-day period for the computer-supported meetings, and limited to a single day for the manual meetings.

Total Meeting Time: The manual meetings lasted an average of 50 minutes, with 20% spent on non-task related activities. The CAIS meetings averaged around 66 minutes, with an un-determinable percentage of time spent on non-task related activities. This total is derived from the summation of all CAIS sessions, measured from the time a user starts up the CAIS software until the user quits. We believe that the additional time spent in the asynchronous meeting is due to four factors: (1) Typing is generally slower than speaking, (2) The reading speed from the screen is about 30% slower than the reading speed from the paper [24, 34], (3) In asynchronous collaboration, a participant is required to read the previous comments for each fault every time to familiarize herself with the context of the discussion up to that point before contributing new ideas, and (4) Since the participants had the freedom of working from home in a relaxed environment, they spent some time checking e-mail, talking to family members, and the like. This time was included as part of the reported time for the asynchronous meetings. Further studies are needed to understand the time requirements of asynchronous meetings better.

Visitation Schedules: Visitation schedules ranged from early morning to late evening times. Most participants preferred to visit CAIS consistently around one time period, generating a unique meeting pattern. Figure 5 plots the time of day against the number of visits in our pilot. Mid-morning and early evening show the highest traffic, while early morning and late nights the lowest.

Post-Manual Questionnaire: Participants were generally satisfied with the manual meeting and agreed with its structure. Scheduling a diverse work group was found to be a difficult task, requiring many rounds of negotiation before a common time was found. The amount of time spent on non-task related issues was small and limited to the exchange of pleasantries at the start and occasional friendly remarks during the meeting.

Post-CAIS Questionnaire: Participants were generally satisfied with the CAIS experience and were not hindered by their distribution across time. They found the CAIS meeting structure acceptable and were able to maintain their train of thought from one CAIS meeting to the next. CAIS matched the participation schedule of each individual participant, allowing them the freedom to meet at a time of their choosing. The notifica-

tion sub-system helped the participants in coordinating their visits, but most participants preferred fewer and less detailed messages.

5.5 Lessons Learned

This sub-section describes the lessons learned in our experience with CAIS.

Increased Time Quantum: CAIS allows the meeting participant as much time as they need to complete their thoughts. In a live meeting, time can be restricted by variables out of the participants control, such as a meeting time limit or being cut off by another participant. In CAIS, thoughts can be composed, edited, deleted, and committed, while in the live meeting, time may be cut short, and thoughts cannot be as easily edited.

HyperText Features: An advantage to having the target material document on-line in the CAIS meeting from the individual fault review was the hypertext link from the comments in the meeting to the document under review. Upon selecting a fault or comment, the browser would automatically scroll and hi-light the line in question. Even compared to well organized handouts for the live meeting, the hyperlinks were viewed as an advantage.

Notification: The notification sub-system was found to be an essential component of our asynchronous collaboration. The notifications were helpful to the participants in determining whether it was appropriate for them to visit the meeting. The micro messages went unread by some, perhaps because the context of the information was lost since the message included only the data entered by the previous participant, and not the context it was addressing. The more brief and concise macro and meta messages were the most appreciated. The participants preferred a single bulk message capturing all three types of information, with less emphasis placed on detailed information.

CAIS As a Complement to Face-to-Face Meeting: Under CAIS we were able to resolve most faults, but some faults could not be resolved asynchronously and were sent to the synchronous meeting. We believe that CAIS can effectively reduce a significant portion of the synchronous part of software inspection, but will not replace it completely.

Participation Schedules and Styles: CAIS tailored well to the participation schedules of all participants. Our non-student group members appreciated being able to attend meetings from their workplaces or homes. Other than having scheduling conflicts people are different in the manner they work. Some find it difficult to concentrate on a task for an extended period of time and opt for brief and more frequent periods of activity instead. Others may wish to work continuously
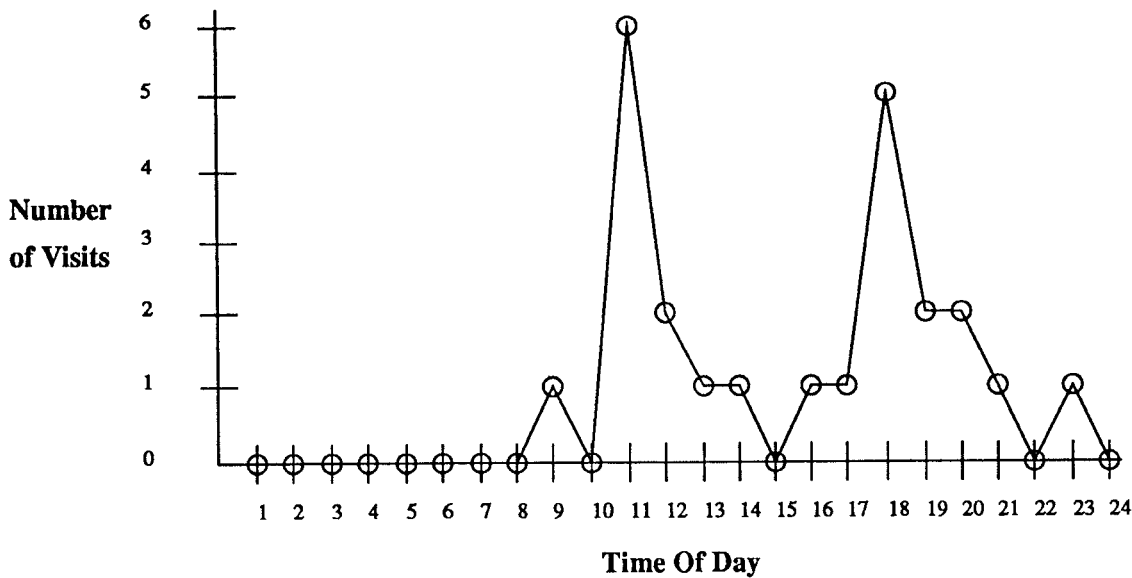
Figure 5: Frequency of Visits

on a task until they are satisfied with their progress. CAIS accommodates both participation styles equally.

CAIS Versus Face-to-Face Meeting: Both reviewers and producers were satisfied with the CAIS experience. The structure imposed by the CAIS meeting was deemed appropriate and effective for software inspection. However, some participants viewed the manual meetings as an easier forum to reach a conclusion for some of the faults. Faults that were poorly-understood had discussions that wandered and were more easily dealt with in a face-to-face meeting. Overall, the participants preferred CAIS over the manual meeting for its flexibility and freedom of meeting times.

6 Conclusions

In this paper, we have introduced asynchrony as a new abstraction in the domain of software engineering and have applied it to a representative software engineering task, namely software inspection. We have assessed the feasibility of asynchrony for software inspection by designing, developing, and conducting a pilot study of an environment for asynchronous software inspection. Through capture and display of a structured history of participants' actions, CAIS effectively supports holding inspection meetings asynchronously. CAIS supports the different schedules and styles of the participants. Whether members prefer to work at day or night, or prefer to put all of their energy into one section of the meeting, CAIS allows everyone to work as they wish. The information management gains, coupled with the hypertext features, are an improvement over the standard document shuffling that occurs in a paper-based meeting.

We put forth CAIS as a complement to the face-to-face meeting, and not as its substitute. Humans are naturally gregarious, and asynchronous meetings take away some of the interplay of a live meeting (jokes, raised eyebrows, etc.). This sort of interaction might not be available to those participating in electronic meetings. By introducing asynchronous meetings to the workplace, just as introducing other types of groupware, our work culture and organization will change. Additional research is needed to understand the effects of the new meeting model on organizations.

We suggest a rich set of problems for future work, based on our experience with CAIS:

Roles: Our current implementation of CAIS does not support the notion of roles, as prescribed by Humphrey [25]. An inspection could assign one of three roles to the participants: Moderator, reviewer, or producer. One application of roles is in the process of group decision making, where the moderator may decide to send a fault to a synchronous meeting after a prolonged asynchronous discussion of the fault has not resulted in consensus amongst participants.

Threaded Comments: CAIS maintains a strict ordering of comments in a discussion by timestamping each comment and keeping the comments in the order of their capture on the time axis. However, independent of its temporal ordering, a comment can be placed after the comment that caused it to happen, indicating a causal relation between the two. For instance, if a participant reads a comment and wishes to respond to it, the reply comment should be placed after the comment the caused it to be recorded, and not at the end

of the comment list as the temporal ordering would dictate. An improvement to our present design is addition of threads to CAIS discussions to better capture the causal ordering of comments, and not merely their temporal ordering.

Inspection Applied to Other Phases of Software Development Waterfall Model: Our efforts this far have included inspection of documents during the requirements and coding phases of their development. Further work is needed in inspecting design documents, such as flow charts and CAD drawings, to better understand the inspection process as applied to other phases of software development waterfall model. We suspect that applying inspection to an earlier phase will prevent the propagation of errors to later phases and will cause a reduction in total development costs.

A Guideline for CAIS Usage: Our pilot studies revealed that approximately 15% of the total faults were difficult to resolve asynchronously and were set aside for a synchronous meeting. Perhaps a set of guidelines could be developed to assist in distinguishing and filtering out such faults prior to the asynchronous meeting. Our preliminary analysis suggests that the category of a fault and the number of messages to be good indicators of such faults. All the faults that were sent to the synchronous meeting were categorized as "major" by the reviewers and involved five or six rounds of messages, without bringing the reviewers any closer to a consensus.

User Studies: We have focused on comparing asynchronized, computer-augmented inspection with face-to-face, paper-based inspection in an effort to study CAIS's feasibility when compared to traditional inspection. A more challenging experiment would be to compare and contrast asynchronized inspection with other computer-augmented inspection meeting modes.

Acknowledgements

References

[1] S.R. Ahuja, J. Ensor, and D. Horn. The Rapport multimedia conferencing system. In *Proceedings of Conference on Office Information Systems*, March 1988.

[2] Marc Andreesen. NCSA Mosaic technical summary. Technical report, University of Illinois, May 1993.

[3] L.M. Applegate, B.R. Konsynski, and J.F. Nunamaker. A group decision support system for idea generation and issue analysis in organization planning. In *Proceedings of the First Conference on Computer-Supported Cooperative Work*, pages 16–34. ACM, December 1986.

[4] Sara A. Bly, Steve R. Harrison, and Susan Irwin. Media spaces: Bringing people together in a video, audio, and computing environment. *Communications of The ACM*, 36(1):28–47, Jan 1993.

[5] B. Boehm. Industrial software metrics top 10 list. In *IEEE Software*, September 1987.

[6] N. Borenstein and N. Freed. MIME: Multi-purpose Internet Mail Extensions. RFC 1521.

[7] N. Borenstein and C. Thyberg. Power, ease of use, and cooperative work in a practical multimedia message system. *International Journal of Man-Machine Studies*, April 1991.

[8] Nathaniel Borenstein. Computational mail as network infrastructure for Computer-Supported Cooperative Work. In *CSCW 92 Proceedings*, pages 67–74, November 1992.

[9] L. Brothers, V. Sembugamoorthy, and M. Miller. ICICLE: Groupware for code inspection. In *Proceedings of Computer Supported Cooperative Work*, pages 169–181, October 1990.

[10] T. Cavalier, R. Chandhok, J. Morris, D. Kaufer, and C. Neuwirth. A visual design for collaborative work: Columns for commenting and annotation. In *Proceedings of HICSS '24 IEEE*, 1990.

[11] J. Conklin and M. Begeman. gIBIS: A hypertext tools for exploratory policy discussion. *ACM Transactions on Office Information Systems*, 6(4), October 1988.

[12] P. Cook, C. Ellia, M. Graf, G. Rein, and T. Smith. Project Nick: Meeting augmentation and analysis. *ACM Transactions on Office Information Systems*, 5(2), April 1987.

[13] Alan M. Davis. *Software Requirements: Analysis and Specification*. Prentice Hall, 1990.

[14] A.R. Dennis, J.S. Valacich, and J.F. Nunamaker Jr. An experimental investigation of the effect of group size in an electronic meeting environment. *IEEE Transactions on Systems, Man and Cybernetics*, 20, 1990.

[15] Gerardine DeSanctis, Marshall Scott Poole, and Gary W. Dickson. Interpretive analysis of team use of group technologies. *Journal of Organizational Computing*, 3(1):1–29, 1993.

[16] P. Dewan and R. Choudhary. Flexible user interface coupling in a collaborative system. *Proceedings of the ACM CHI's 91 Conference*, April 1991.

[17] P. Dewan and E. Vasilik. An object model for conventional operating systems. *Usenix Computing Systems*, December 1990.

[18] Prasun Dewan and John Riedl. Toward computer-supported concurrent software engineering. *IEEE Computer*, Jan 93.

[19] Gary Dickson, Marshall Scott Poole, and Gerardine DeSanctis. *An Overview of the GDSS Research Project and the SAMM System*, chapter 9, pages 163–179. Van Nostrand Reinhold, 1992.

[20] C. Ellis, S. Gibbs, and G. Rein. Groupware: Some issues and experiences. *Communications of the ACM*, pages 39–56, January 1991.

[21] H.C. Forsdick and R.H. Thomas. The design of Diamond: A distributed multimedia document system. Technical report, TR number 5402, Bolt Beranek and Newman Inc., October 1982.

[22] G. Forte and R.J. Norman. A self-assessment by the software engineering community. *Communications of the ACM*, 35(4):28–32, April 1992.

[23] John W. Gintell and Gerard Memmi. CIA: Collaborative Inspection Agent experience: Building a CSCW application for software engineering. In *Workshop on CSCW Tools*, October 1992.

[24] J.D. Gould and N. Grischkowsky. Doing the same work with hard copy and with cathode ray tube (CRT) computer terminals. In *Human Factors*, pages 323–337. ACM, 1984.

[25] W.S. Humphrey. *Managing the Software Process.* Addison Wesley, 1989.

[26] C.M. Hymes and G. Olson. Unblocking brainstorming through the use of a simple group editor. In *CSCW 92 Proceedings*, 1992.

[27] Simon M. Kaplan, William J. Tolone, Douglas P. Bogia, and Celsina Bignoli. Flexible, active support for collaborative work with ConversationBuilder. In *CSCW 92 proceedings*, November 1992.

[28] K.Y. Lai and T.W. Malone. Object lens: A spreadsheet for cooperative work. In *Proceedings of 1988 Conference on Computer Supported Cooperative Work*, 1988.

[29] Lotus Development Corporation. *Lotus Notes: The Groupware Standard*, release 3 edition, 1993.

[30] T. Malone, K. Grant, F. Furback, S. Brobst, and M. Cohen. Intelligent information-sharing systems. *CACM*, 30(5):390–402, May 1987.

[31] Vahid Mashayekhi, Janet Drake, Wei-Tek Tsai, and John Riedl. Distributed collaborative software inspection. *IEEE Software*, pages 66–75, September 1993.

[32] C.M. Neuwirth, D.S. Kaufer, R. Chandhok, and J.H. Morris. Issues in the design of computer-supported for co-authroing and commenting. In *Proceedings of the Third Conference on Computer-Supported Cooperative Work*, pages 183–195. Association for Computing Machinery, 1990.

[33] J.F. Nunamaker, A.R. Dennis, J.F. George, W.B. Martz, J.S. Valacich, and D.R. Vogel. *GroupSystems*, chapter 8, pages 143–162. Van Nostrand Reinhold, 1992.

[34] P.Wright and A. Lickorish. Proof-reading texts on screen and paper. *Behavior and Information Technology*, 2(3):227–235, July-September 1983.

[35] John Riedl, Vahid Mashayekhi, Jim Schnepf, Mark Claypool, and Dan Frankowski. Suitesound: A system for distributed collaborative multimedia. *IEEE Transactions on Knowledge and Data Engineering*, pages 600–610, August 1993.

[36] S. Sakata. Development and evaluation of an In-house multimedia desktop coneference. *IEEE journal on selected areas in communcations*, April 1990.

[37] M. Stefik, G. Foster, D.G. Bobrow, K. Kahn, S. Lanning, and L. Suchman. Beyond the chalkboard: Computer support for collaboration and problem solving in meetings. *Communications of the ACM*, 30(1):32–47, January 1987.

[38] Apollo Systems. Introduction to the DSEE environment. User Reference Manual.

[39] J.M. Tazelaar. In depth groupware. *Byte Magazine*, December 1988.

[40] Walter F. Tichy. RCS - A system for version control. *Software Practice and Experience*, 15(7):637–654, July 1985.