A Case Study of Distributed, Asynchronous Software Inspection

Michael Stein, John Riedl
Dept. of Computer Science
University of Minnesota
200 Union St. S.E.
Minneapolis, MN, USA 55455
+1 612 624 {8372, 7372}
{mstein, riedl}@cs.umn.edu

Sören J. Harner
ICEM Systems, GmbH.
Kuesterstraße 8
30519 Hanover, Germany
+49 511 9848 713
sjh@icem.de

Vahid Mashayekhi *
DELL Computer Corporation
2214 W. Braker Lane
Suite D
Austin, TX, USA 78758
+1 512 728 3653
victor_mashayekhi@us.dell.com

ABSTRACT
Traditional software inspection requires participants to meet together at the same time in the same place. Distributed, asynchronous inspection allows participants to conduct meetings independently of time and space, making inspection more convenient. We report on an industrial study that we have performed using a tool designed for distributed, asynchronous software inspection. Our experience suggests that distributed, asynchronous software inspection is feasible, and is a cost-effective means of collaboration for geographically distributed work groups.

Keywords
Concurrent Software Engineering, Software Inspection, CSCW, Collaboration, Groupware, World Wide Web.

INTRODUCTION
Software inspection is a widely practiced, highly structured, collaborative software engineering activity. The traditional model of inspection is centered around a centralized, synchronous inspection meeting. This places both space and time constraints on the inspection.

Participants are constrained in space to be at the site of the inspection meeting. Travel to the meeting site can be costly in terms of both time and money for distributed workgroups. One alternative is to hold an inspection meeting by audio or video conference, with the addition of a shared text space on which everyone can simultaneously view inspection artifacts[14]. However, this alternative does not solve the time constraint described next.

Participants are constrained in time by the need to attend the inspection meeting. Even for co-located workgroups, it may be days between the time the inspection material is ready and the time all inspectors are free to attend a meeting[1]. Additionally, for intercontinentally distributed workgroups, in may even be impossible to schedule an audio or video conference during working hours for all participants.

*This research was conducted while Dr. Mashayekhi was at the University of Minnesota.

Advances in distributed systems, networks, and user interface technology have enabled distributed, asynchronous meetings to become a viable alternative to face-to-face (FtF) meetings[2]. Building on these advances, we built and experimented with various prototype tools for collaborative review. The Collaborative Software Inspector (CSI)[14] provided computer support for synchronous inspection using the Suite[4] infrastructure. The Collaborative Asynchronous Inspector of Software (CAIS)[15] extended CSI to support distributed, asynchronous inspections of text, and was implemented on Lotus Notes[20] as well as Suite.

We have extended these previous prototype tools to provide a World-Wide-Web ("Web") based tool for distributed, asynchronous software inspection of textual and graphical artifacts, which we call the Asynchronous Inspector of Software Artifacts (AISA)[16].

This paper reports our experiences with using AISA for software inspections in a professional development organization split between Germany and two locations in the USA, and for inspection of graphical artifacts.

Hypothesis
Our goal in building and using AISA has been to explore the hypothesis:

> **Distributed, asynchronous software inspection can be a practical method for software inspection of both graphical and textual artifacts.**

Model of Software Inspection
Software inspection is a detailed review of a small amount of material by technically competent peers with the goal of detecting faults [7]. We implement Humphrey's model [12], because it is highly structured and provides intermediate results through individual

and correlated fault lists.

In Humphrey's model, the inspection team is a group of peers with the technical knowledge required for detailed inspection. Participants have specific roles: reviewer, moderator, producer, and recorder. Each participant except the producer prepares for the inspection by looking over the material, creating a fault list, and giving the list to the producer before the meeting. The producer correlates the faults and prepares to address the faults in the inspection meeting. All participants attend the inspection meeting and discuss the faults. The quantity of target material addressed in one inspection is small because of the detailed level of review. The four phases of software inspection are:

Initialization: The target material is identified and the participants are chosen. The producer and moderator may hold an optional introductory meeting. The inspection meeting is scheduled, and criteria for the inspection are chosen.

Preparation: The target material and inspection criteria are distributed to the reviewers. The reviewers examine the target material and create individual fault lists, giving the lists to the producer when their examinations are complete. Then the producer merges the individual fault lists into a correlated fault list.

Discussion: The producer and reviewers discuss the issues on the correlated fault list, commonly in an FtF meeting. When consensus on fault resolution is reached, required actions are recorded on an action item list. Finally, it is decided whether a re-inspection is needed.

Post-Discussion: After the producer fixes the problems by completing each action item, the moderator re-evaluates the target material. If a re-inspection is required, the inspection process begins again.

Thus the inspection process consists of two distinct meeting modes: fault collection and discussion. During fault collection, individuals review the documents independently and are not restricted by place and time. In the discussion, all participants discuss the correlated fault list generated by the producer. The discussion is traditionally a same-time, same-place meeting. We explore the potential for replacing this meeting with a distributed, asynchronous discussion. Effective conduct of such a discussion would allow software inspection to be done efficiently by widely distributed workgroups.

User-level Collaboration Requirements

We have identified the following user-level collaboration requirements for FtF inspections that we wish to conserve in asynchronous inspection:

Threads of Discussion: Participants must be able to conduct discussions on faults during the meeting. In FtF meetings, discussions are pursued serially. Similarly, within a discussion, all comments are made serially, with participants taking turns to voice their opinions.

Sharing of Information: Participants must be able to share inspection information with one another. This information consists of the target material, correlated fault list, in-meeting inspection artifacts, post-meeting summaries, and action items. In a paper-based meeting, this is done by copying the paper artifacts, or displaying them on a screen.

Train of Thought: Participants must be able to maintain their train of thought during the meeting. Train of thought is sustained in FtF inspection meetings, since each meeting is held as a single session from start to finish.

Visual Cues: Participants must be able to direct the attention of other participants to areas of interest in the information space. In FtF meetings, speakers often achieve this goal by using pointers to provide a common focus for the group and guide the participants through the inspection material.

Reaching a Consensus: Participants must be able to arrive at decisions that resolve differences. In FtF meetings, once a discussion has matured, a potential resolution is identified and is placed before the group as a proposal. The participants must decide whether they collectively agree with a proposal either formally (by voting) or informally.

Coordination: Participants must be able to communicate with one another to coordinate their activities and ensure that the group's objectives are met. In FtF meetings, participants use verbal communication for moving through the meeting agenda and ensuring that project deadlines are satisfied.

Inspection History: A record of the inspection must be kept, identifying issues raised and their resolutions. This record must be persistent, so that it can be used throughout the development cycle for technical reference (e.g., a design history) and for process improvement. In an FtF meeting, this record is kept on paper, and possibly moved onto electronic media after completion of the inspection.

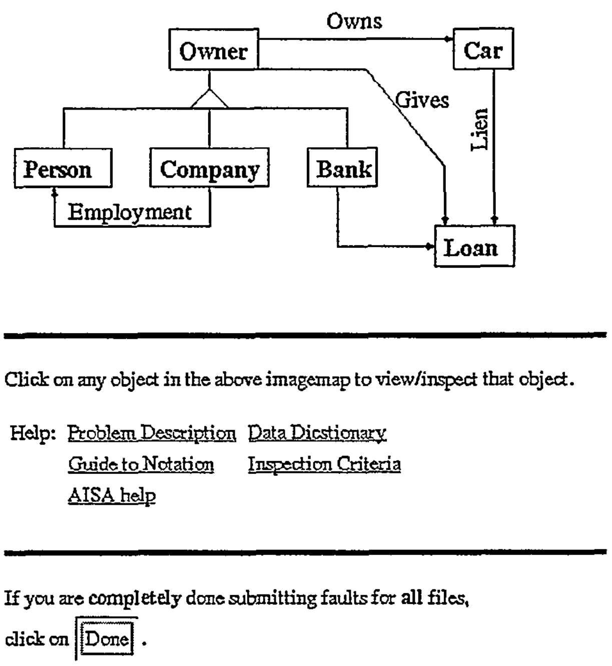
Asynchronous Tool Design Elements

To meet the set of requirements above, we have identified three basic design elements for distributed, asynchronous inspections:

Shared Information Space: The shared information space plays a pivotal role in asynchronous inspection. It presents a causal and temporal ordering of the inspection activities, organizes the target material into a hierarchy of its logical parts, captures and organizes the additions participants make to the information space, groups interrelated faults as a single composite fault, controls access to the information, and ensures that the inspection material remains available after the conclusion of the inspection. The shared information space satisfies the "Sharing of Information", "Visual Cues", and "Inspection History" requirements.

Group Decision Support: Once a proposal is put forth by an inspection participant, group members need to decide its status. A tool for asynchronous inspection must support these activities. Group decision support satisfies the "Reaching a Consensus" and "Coordination" requirements.

Communications: In an asynchronous meeting, participants must be made aware of what others are doing, or what they are expected to contribute toward task completion. This awareness can help prevent duplication of efforts, coordinate the group activities, and ensure project deadlines are met. For instance, during the discussion phase of the inspection, participants must be informed that the discussion has begun for various correlated faults, they must be able to view the ongoing discussions, and they must be informed that an issue is resolved. Communications satisfies the "Thread of Discussion", "Train of Thought", and "Coordination" requirements.



Click on any object in the above imagemap to view/inspect that object.

Help: Problem Description Data Dicstionary
Guide to Notation Inspection Criteria
AISA help

If you are completely done submitting faults for all files,
click on Done .

Figure 1: *Sample home page for graphical inspection.*

RELATED WORK

Research in the area of software inspection has resulted in the introduction of computer-supported tools. Such tools include ICICLE, CIA, CSRS, and Scrutiny, as well as our tools CSI and CAIS (described above).

ICICLE [3] is a system intended to support the set of tasks performed during code inspection. ICICLE assists individual users in the comment-preparation phase of code inspection. It provides a synchronous environment in the inspection meeting phase, with computer support providing a paperless meeting.

Collaborative Inspection Agent (CIA) is a document inspection tool [10]. CIA supports synchronous inspection of all work products at various stages of the life cycle. It supports collaborative work by simultaneously displaying information on multiple users' screens, and allowing participants to play inspection roles.

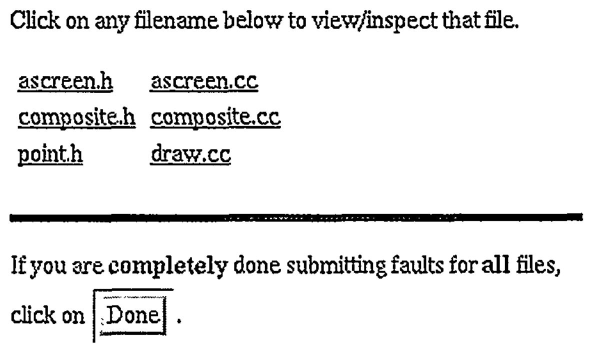
Collaborative Software Review System (CSRS) [13] aims to decrease the required human effort in reviews, conduct inspections incrementally during the software development, and provide on-line capabilities to collect metrics on the inspection process and software artifacts. The system is implemented on top of EGRET, a multi-user, distributed, hypertext environment for asynchronous collaboration.

Scrutiny [9] is a collaborative inspection system that has been successfully used for professional software development. It supports reviewers in a synchronous meeting. Lotus Notes could also be used to support software inspection of textual artifacts [20].

AISA goes beyond ICICLE, CIA, and CSI by supporting asynchronous, distributed inspections. AISA goes beyond Scrutiny, CSRS, and CAIS by supporting inspection of graphical documents. AISA, alone among the systems we have examined, is built upon the Web infrastructure.

AISA INSPECTION SCENARIO

This section describes how an inspection would be con-

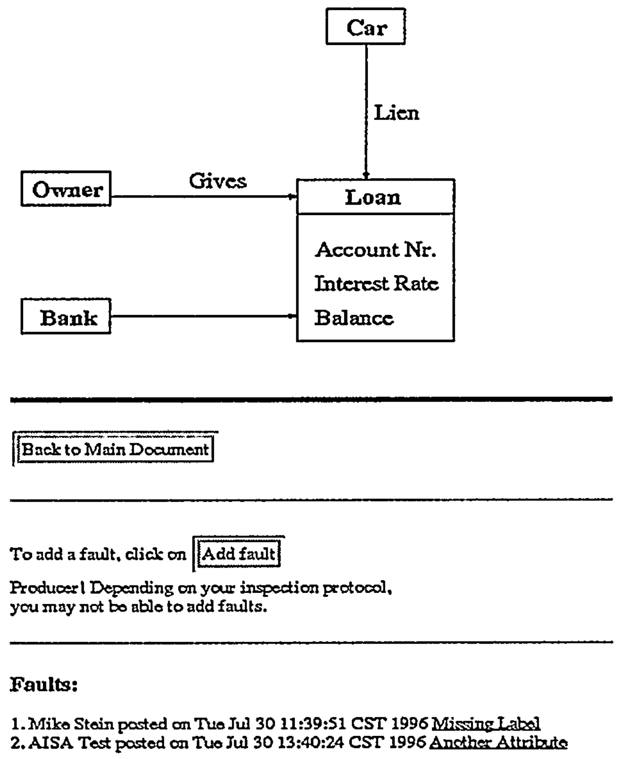


Click on any filename below to view/inspect that file.

ascreen.h ascreen.cc
composite.h composite.cc
point.h draw.cc

If you are completely done submitting faults for all files,
click on Done .

Figure 2: *Sample home page for textual inspection.*



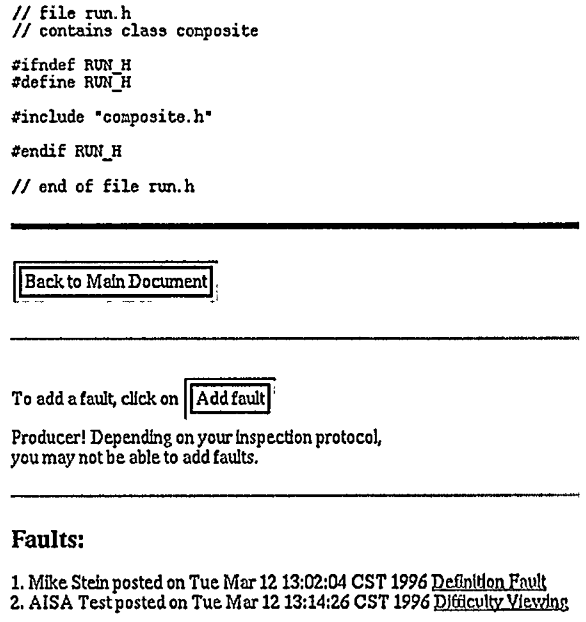
To add a fault, click on Add fault

Producer! Depending on your inspection protocol, you may not be able to add faults.

Faults:

1. Mike Stein posted on Tue Jul 30 11:39:51 CST 1996 Missing Label
2. AISA Test posted on Tue Jul 30 13:40:24 CST 1996 Another Attribute

Figure 3: *Graphical artifact to inspect.*



```
// file run.h
// contains class composite

#ifndef RUN_H
#define RUN_H

#include "composite.h"

#endif RUN_H

// end of file run.h
```

To add a fault, click on Add fault

Producer! Depending on your inspection protocol, you may not be able to add faults.

Faults:

1. Mike Stein posted on Tue Mar 12 13:02:04 CST 1996 Definition Fault
2. AISA Test posted on Tue Mar 12 13:14:26 CST 1996 Difficulty Viewing

Figure 4: *Textual artifact to inspect.*

ducted using the Web-based AISA tool. We describe the use of the tool during the four phases of software inspection described previously.

Initialization Phase

The moderator and producer decide what material is to be inspected, and break large software artifacts into multiple inspections if necessary. The artifacts may be graphical or textual. Then the moderator decides the *granularity* of fault collection on the material: during the fault collection phase, reviewers will attach faults to particular inspection items, and the moderator must set up the inspection as a set of items, to each of which reviewers may attach faults.

Thus, for an inspection of C++ classes, the granularity for fault collection might be a single class. For inspection of a graphical artifact, the granularity might be a single graphical object, or a small set of such objects.

The moderator puts the material in a node that is known to the web server, and creates a home page for the inspection (Figures 1 and 2). Since there is no scheduled FtF meeting the Moderator must also set the schedule for completing fault collection, fault correlation, and discussion.

Preparation Phase

The moderator announces the Uniform Resource Location (URL) of the inspection's home page to the participants. The reviewers examine the target material on-line, or by printing paper copies (Figures 3 and 4).

When reviewers identify faults, they click on the "Add Fault" button to go directly to the fault collection template, onto which they add faults. (Figure 5).

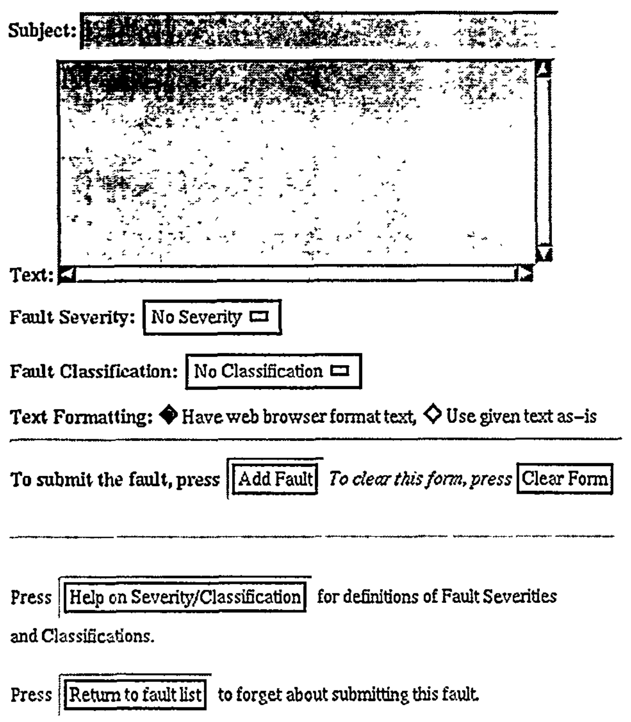
There are two methods of identifying the reviewer who added a fault. Automatic identification is provided using the "identd" daemon, but this doesn't always work when participants are behind proxy servers (as in our industrial inspections). In this case, reviewers identify themselves manually.

After completing inspection of the target material, the reviewer presses the "Done" button on the home page, and waits for the producer to correlate faults.

Once all reviewers have finished submitting faults, the producer performs Fault Correlation. The producer examines each fault, and groups duplicated faults together into a single "merged" fault (Figure 7). This is a two-phase process. First, the producer looks at each artifact, and groups all duplicated faults for that artifact (Figure 6). Then the producer looks at the concatenation of these per-artifact correlated lists, and groups identical faults that were submitted to different artifacts; an example of this would be when two reviewers assign an interface fault to artifacts on either side of the interface.

Discussion Phase

A distributed, asynchronous meeting is carried out through the discussion phase of the inspection. The


Subject:

Text:

Fault Severity: No Severity

Fault Classification: No Classification

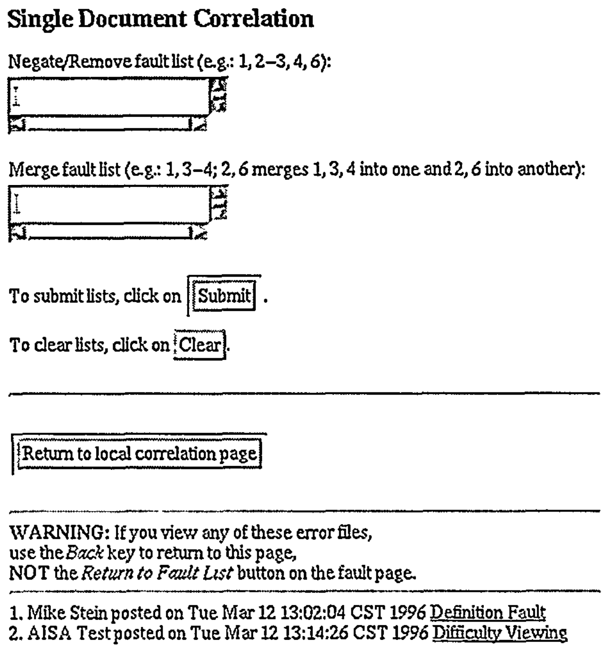
Text Formatting: ◆ Have web browser format text, ◇ Use given text as–is

To submit the fault, press Add Fault *To clear this form, press* Clear Form

Press Help on Severity/Classification for definitions of Fault Severities and Classifications.

Press Return to fault list to forget about submitting this fault.

Figure 5: *Reviewer's fault collection template.*


Single Document Correlation

Negate/Remove fault list (e.g.: 1, 2–3, 4, 6):

Merge fault list (e.g.: 1, 3–4; 2, 6 merges 1, 3, 4 into one and 2, 6 into another):

To submit lists, click on Submit.

To clear lists, click on Clear.

Return to local correlation page

WARNING: If you view any of these error files,
use the *Back* key to return to this page,
NOT the *Return to Fault List* button on the fault page.

1. Mike Stein posted on Tue Mar 12 13:02:04 CST 1996 Definition Fault
2. AISA Test posted on Tue Mar 12 13:14:26 CST 1996 Difficulty Viewing

Figure 6: *Producer's fault correlation template*

AISA tool recognizes that the producer has completed fault correlation, and sends e-mail to all participants indicating that the discussion has begun. When participants next access the inspection's home page, they are led to a list of faults for discussion.

Participants can view a fault, comment on it, and submit a proposal for resolution of the fault. When a proposal has been submitted, reviewers (but not the producer) vote on it electronically. Voting can be set up to require either unanimous agreement for approval, or a majority vote. If a proposal is accepted, the discussion for this item is closed, and the item is marked as resolved. Otherwise, discussion continues.

All participants can view all comments made using the discussion page for a given fault, which holds a threaded list of discussion items. If the moderator decides that an issue is unresolvable within the context of the discussion, the moderator may mark the fault as tabled for future analysis, and disallow further discussion of this fault within the confines of the inspection (Figure 7).

Post-Discussion Phase

The Moderator closes the inspection after all issues are resolved, or no further progress is being made. AISA automatically sends the participants notification that the inspection is closed. Then the Moderator works with the Producer and appropriate reviewers to resolve any issues that were left for later resolution.

Inspection artifacts are retained by AISA, so they can be used for historical purposes and data analysis.

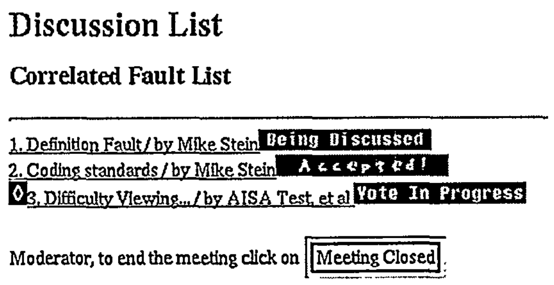
INSPECTION EXPERIENCE

We have done two types of inspections with AISA. We inspected C++ class definitions for an ICEM Systems class library, to study distributed, asynchronous inspection in an industrial setting. The original industrial trial consisted of two inspections. However, owing to the positive experiences ICEM had with these inspections, they conducted the remaining inspections of the class definitions with AISA. Our data cover five inspections of artifacts involved in base functionality for commercial products.

We also inspected graphical object diagrams at the University of Minnesota with graduate students to demonstrate feasibility of graphical inspection.

Measurements Taken

AISA collects the following metrics from each inspection phase: (1) *Fault Collection:* Number of faults recorded by each participant; (2) *Fault Correlation:* Number of duplicates removed and number of faults merged; (3) *Inspection Meeting:* Number of comments per person on each fault, number of proposals per person, number of votes per person, number of resolved faults, and number of unresolved faults. AISA also maintains a visitation schedule for each participant (i.e., AISA timestamps and records each access to a Web page).


Discussion List

Correlated Fault List

1. Definition Fault / by Mike Stein Being Discussed
2. Coding standards / by Mike Stein Accepted!
3. Difficulty Viewing... / by AISA Test, et al Vote In Progress

Moderator, to end the meeting click on Meeting Closed.

Figure 7: *Discussion screen*

Additionally, we measured qualitative characteristics by asking the subjects to complete questionnaires (on the Web) after the inspection. We inquired about their degree of satisfaction with the inspection experience, level of agreement with the inspection structure, level of flexibility provided in their participation schedule, usefulness of the notification messages, suitability of the decision making methods, degree of participation in the discussions, and meeting preferences. [1]

Overview of Textual Inspection

The target material in the ICEM inspections was a set of C++ class definitions for a corporate class library. The goals of the inspections of textual artifacts at ICEM Systems were: (1) to test the feasibility of AISA in an industrial setting, and (2) to enable ICEM Systems to inspect software artifacts with a distributed workgroup in a cost-effective manner.

Participants were distributed among 3 locations in Germany, and two in the United States (in Minnesota and Ohio). They used a variety of Web browsers (Mosaic and Netscape graphical browsers, and the Lynx text-only browser) to access the material from both their offices and homes. These inspections used manual identification, because the German participants accessed the material from behind proxy servers.

Textual Inspection Results

[1]All measurements were used for study purposes only. Raw results were not shared with management.

Inspection Nr.	1	2	3	4	5
Classes	10	20	4	8	10
Member Functions	113	317	49	131	231
Size (text bytes)	5359	15364	4580	8200	8487
Participants	5	6	6	5	6
Faults Collected	22	56	38	54	64
Merged Faults	14	50	15	32	40
Discussion Items	25	22	32	46	59
Faults Resolved	6	16	15	16	34

Table 1: Table of inspections at ICEM Systems.

Inspection Nr.	1	2	3	4	5	Mean	SD
Member Functions	113	317	49	131	231	168	106
Fault Duplication	1.6	1.1	2.5	1.7	1.6	1.5	0.7
Fault Density	.12	.16	.31	.24	.17	.20	.07
Discussion Activity	1.8	0.4	2.1	1.4	1.5	1.5	0.9
Fault Resolution	.43	.32	1.0	.50	.85	.62	.29

Table 2: Results from ICEM Systems inspections

Table 1 summarizes the raw results for the inspections of the C++ class definitions. These were inspections of detailed design information, not of source code. In the following tables and figures, we denote these as inspections 1 through 5.

Size measures available to us for the amount of material reviewed were the number of classes, the number of member functions, and the total documentation size in bytes. The documentation size was unreliable because the size includes hyperlinks and other formatting artifacts. The number of member functions reviewed is more informative than the number of classes, since the classes were of very different sizes. So the number of member functions was our measure of inspection size.

An inspection is effective if participants find the faults in the material being inspected, and are able to resolve them to their mutual satisfaction. We have identified the following ratios as measures of inspection effectiveness. We report their values in Table 2, and we discuss them below.

Fault Duplication: The ratio of Faults Collected to Merged Faults gives the mean number of times the same fault was reported by different people. Thus a high ratio indicates that many of the faults were discovered by more than one person. [2]

Fault Density: The rate of Merged Faults per Member Function indicates the fault density of the material. A low Fault Density may signify either an excellent design, or a cursory inspection of the material.

Discussion Activity: The number of Discussion Items per Merged Fault measures the amount of activity that took place during the discussion phase. A high value suggests an active discussion.

Fault Resolution: The percentage of Merged Faults resolved within the AISA asynchronous inspection is

[2]Reviewers could view each others' faults as they worked, so some people may have discovered a fault, observed that others had already entered it, and not entered a duplicate.

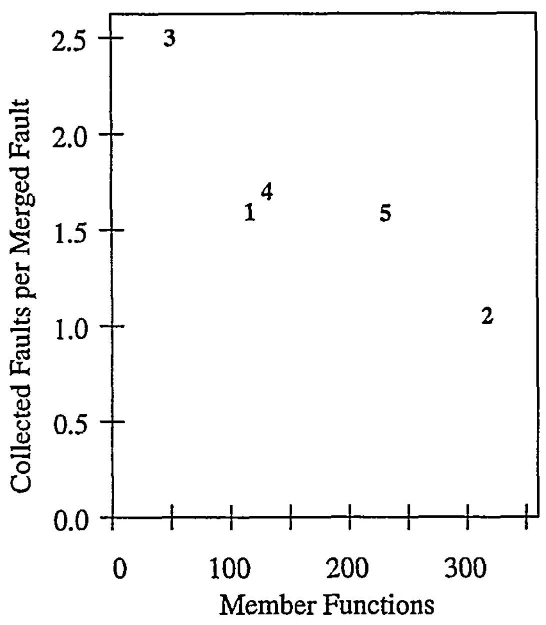


Figure 8: *Fault Duplication*

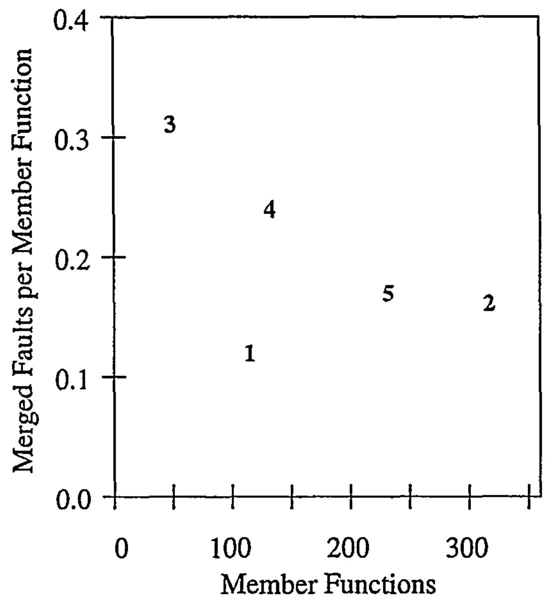


Figure 9: *Fault Density*

the key measure of effectiveness. For asynchronous inspection to be *completely* effective, all issues must be resolvable without traditional meetings.

In analyzing these results, we were interested in trends *within* the inspections studied. The raw values of these metrics are of little use outside this context, because they might be expected to differ for different organizations, different types of software products, and artifacts at different phases of the development process.

Fault Duplication
There was wide variation in the average number of reviewers who identified the same fault, from 1.1 to 2.5 reviewers. Figure 8 shows a discernible trend toward more people independently finding the same fault for smaller inspection sizes.

Fault Density
The density of merged faults per member function was relatively consistent throughout the five inspections, with a mean of 0.20 faults/function and a standard deviation of 0.07 (Figure 9). There was no clear trend in the fault density as a function of inspection size.

Discussion Activity
The number of discussion items per fault was very low in inspection 2 (0.4), but had less variation among the other inspections. Inspection 3 had the most discussion activity (2.1 items/fault) (Figure 10). Discussion activity tended to drop off with increasing inspection size.

Fault Resolution
Percentage of faults resolved in the meeting ranged from 32% (inspection 2) to 100% (inspection 3), with a wide variation among inspections (Figure 11), but no definite trend toward greater fault resolution with smaller inspection size.

However, even in the absence of trends for some of the metrics we used, inspections 2 and 3 had the outlying points for these metrics , and in opposite directions. We discuss the meaning of this in Lessons Learned.

Overview of Graphical Inspection
At the University of Minnesota, we inspected a graphical object diagram of the workings of an automatic teller machine. The goals of this inspection were to test the feasibility of AISA for inspecting graphical artifacts, and to measure the efficacy of asynchronous inspection in locating faults. As part of our test, we seeded the artifacts with 10 faults. If the inspectors could not find most of these faults, we felt the inspection could not have been effective.

Participants were four graduate students in the Computer Science department at the University of Minnesota, including two of the authors. All participants had at least 1.5 years of industrial software development experience. Participants were all located at the university, and accessed the inspection material via a local area network. They were identified automatically by the "identd" daemon.

Graphical Inspection Results
The three reviewers recorded 28 total faults, with each reviewer discovering either 9 or 10 faults. 7 of the 10 seeded faults were found. The producer merged these faults into 14 distinct faults. The discussion of these 14 faults resulted in 25 discussion items. Resolutions were

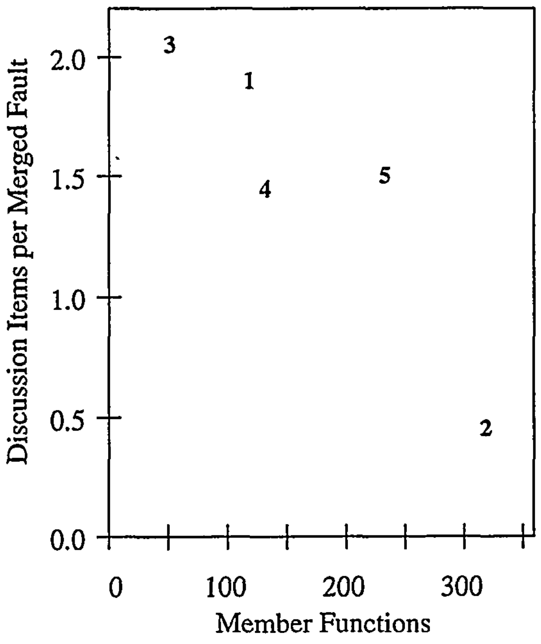


Figure 10: *Discussion Activity*

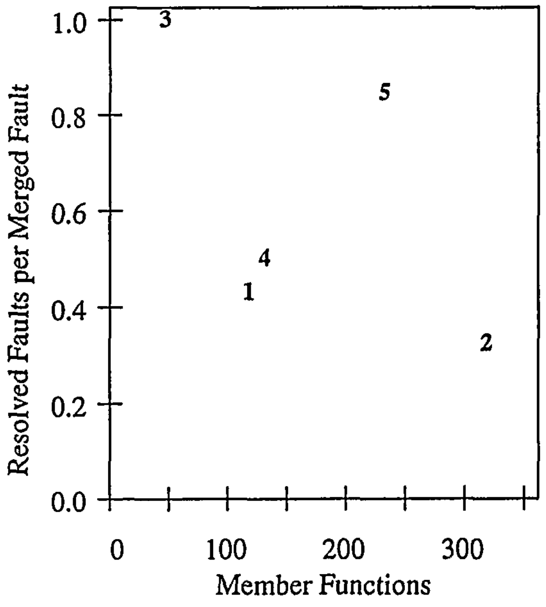


Figure 11: *Fault Resolution*

proposed for 12 faults, and all were accepted.

LESSONS LEARNED

From using the AISA tool, we learned lessons about distributed asynchronous inspection, and about the use of the Web as a collaborative infrastructure. Lessons we learned were derived from questionnaires, e-mail messages, and discussions with inspection participants.

Distributed, Asynchronous Inspection in the Field

Practicality

Both professional developers and university students reported that the AISA tool was useful. The professional developers at ICEM Systems have used AISA for nine more inspections since we collected our data. The graduate students were able to successfully inspect the graphical artifacts, finding 7 of 10 seeded faults. [3]

The learning curve for new AISA users was short. It commonly took ICEM participants one inspection to learn to use the tool. They were effective in using AISA the second time they used it.

Organization and Division of Material

Consistently, Inspections 2 and 3 produced the data furthest from the mean values for the inspections. Inspection 2 had the fewest faults found by more than one reviewer, the fewest discussion items per fault, and the lowest rate of fault resolution within the meeting of any inspection undertaken. Inspection 3 had the most faults found by multiple reviewers, the most discussion items per fault, and resolved all faults discovered.

The artifacts for both inspections 2 and 3 were written by the same experienced author, and they followed each other chronologically in time, with many of the same reviewers (already trained during inspection 1).

We hypothesize that the above differences are largely due to inspection size. Inspection 2 was the largest inspection (317 functions), and inspection 3 was the smallest (49 functions). Participants commented that inspection 2 contained too much material for them to deal with. Inspection 3 was intentionally made small.

Synchronous inspections normally cover a limited amount of material, so that the meetings remain short. It might be hypothesized that asynchronous inspections would have no such limitation, because such inspections don't require a meeting and allow people to perform inspection activities at their convenience. But our results suggest that even asynchronous inspections have size limitations.

AISA also allows all participants to view everyone's collected faults at all times. Other practitioners have prevented reviewers from seeing each others' faults while they are entering their own faults, on the theory that people would not enter faults they thought would duplicate faults already entered[13]. We have found that if the inspection is of manageable size, many people would enter duplicate faults, anyway. So fault collection does not appear to be compromised.

Decision-Making

Two closely-related issues in decision-making are: (1)

[3]We considered this successful because the students were not experts in the problem domain.

What issues are resolvable asynchronously? and (2) What is the best way to resolve them? As we add complexity to the decision support system, more faults can theoretically be resolved asynchronously, but resolving the simple faults becomes harder.

For four of the five ICEM inspections, some issues had to be resolved after the asynchronous inspection ended. The moderators noted that the issues that needed post-inspection resolution meetings tended to be issues where different parties had to compromise to reach agreement, or where the author or a reviewer lacked the understanding to come to a proper fault resolution.

The use of voting for decision support caused the most difficulties with asynchronous inspection. Originally, AISA was designed to suspend discussion while a vote was being taken, to prod people to vote on an issue. The graduate students found it was important that the discussion continue even when a proposal is outstanding, so we added that before the ICEM inspections.

Even so, ICEM reviewers were dissatisfied with voting, and questioned the need for a formal voting mechanism. Essentially, they felt that the voting procedure was cumbersome for indicating agreement with simple fault corrections, yet not capable of supporting decision-making for complex issues.

There must be a lightweight method for reviewers to indicate agreement with resolutions of simple faults (such as "the second parameter to function X should be a long integer, not a short integer"). It appears complex issues may require post-discussion activity regardless of decision support mechanism (even regardless of meeting type [17]), so further research is needed before implementing a rich, complex decision-support structure.

Depth of Discussion
We found that the "depth" of the comments made in asynchronous inspection was greater than that normally seen in inspection meetings. We feel this is because participants have a chance to compose their thoughts before responding to comments, instead of having to respond immediately as in a meeting. We especially noted the following types of thoughtful discussion items.

Rebuttal: The author uses detailed examples to show how an hypothesized fault is, in fact, correct.

Examples: A reviewer shows why a certain design leads to a fault, or how a certain fault resolution solves a problem.

Citation: A participant refers to a publication or a previous design that they feel should be considered before the discussion moves ahead. A useful variant on this is to reference an on-line Web source, and to embed the link in the comment.

Development History
An idea formulated clearly in text becomes part of the knowledge base; it can be archived and reused. Documenting designs through asynchronous threaded discussions can serve as a parallel communication channel that can be used in concert with more traditional design documentation to provide better understanding, and to enhance traceability of development information [11].

Asynchronous vs. Synchronous Inspection
Size of Inspection
We found that even in an asynchronous inspection where meeting time was not an issue, the amount of material to be reviewed must be kept small. Inspection 2 was too large; participants felt overwhelmed by the amount of material.

Roles of the Participants
As happens in synchronous inspections, supporting the notion of roles in AISA divided the work and responsibility in our studies. The greater authority given to the moderator role in AISA came especially into play in the ICEM inspections, where the moderator was truly an expert on the subsystem under inspection.

Experiences of the Participants
Participants in all inspections found AISA easy to use after a short learning curve. They appreciated being able to work at a time of their choosing. One participant also appreciated being able to use a a text-only (Lynx) browser.

Unfortunately, the response time was found to be slow at times, especially on intercontinental links. A moderately fast Internet connection is clearly important for participant satisfaction.

Cost-effectiveness
AISA makes inspection possible for distributed workgroups in cases where inspection was impossible before because of the costs of airfare and travel time, or cost and clumsiness of holding a meeting by conference call.

Although we have not formally studied the relative costs of synchronous vs. asynchronous inspection for collocated workgroups, we observe the following points concerning cost-effectiveness of synchronous vs. asynchronous inspection:

- Fault collection with AISA takes about the same time as it would for a traditional inspection, because the procedure is similar. In fact, people sometimes did fault collection by printing hard copies of the material to review.
- Any items that can be resolved asynchronously reduce the time of any meeting, and possibly reduce the number of participants needed at that meeting. This cuts costs by improving time to market,

since it can take days before an inspection can be scheduled [1].

The Web as a Collaborative Infrastructure

We discovered the Web to be useful as a collaborative infrastructure. Its greatest advantages that we saw were:

Availability: The Web is widely available, making our tool available to those who wish to try it. AISA is simple to add to a corporate intranet, and to use remotely (security permitting).

Portability: AISA can be used on any Web server supporting CGI Web scripting and Perl. All setup and installation is on the server side. Any client workstation with a Web browser that can access the server can be used. To use automatic identification, the client and server must both run "identd".

Familiarity: The Web's familiarity to the participants helped make the AISA learning curve short. People didn't need to learn any new infrastructures with which they were unfamiliar. This familiarity also allowed people to make clever use of its features, for instance by embedding links in their comments.

But the Web is deficient in features found in some other collaborative infrastructures on which we have implemented a similar inspection tool. Lotus Notes has better security and support for replication[20]. Suite[4] supports synchronous and asynchronous coupling, and better automatic generation of collaborative user interfaces.

We were able to implement AISA, or a close approximation to it, on various infrastructures. We used the Web for industrial inspections because of its ubiquity and graphics support.

CONCLUSIONS

Distributed, asynchronous software inspection relaxes the constraints that all participants work together at the same time and in the same place. We have designed and implemented AISA, a tool for distributed, asynchronous inspection. We have inspected graphical artifacts in a university setting, and textual artifacts in an industrial setting with participants distributed across continents. We use the results from these inspections as evidence to evaluate the hypothesis that we introduced earlier:

> **Distributed, asynchronous software inspection can be a practical method for software inspection of both graphical and textual artifacts.**

Our results support this hypothesis. We have developed a tool that supports distributed, asynchronous software inspection. We have used this tool to successfully inspect both graphical and textual software artifacts. We were able to successfully inspect the textual artifacts using an inspection team spread among four locations on two continents, with a seven hour time difference among them. Participants and their management liked the tool so well that they continued to use it for other work after the study ended.

Distributed, asynchronous inspection removes the constraints that the participants work together at the same time and place. It also gives everyone a chance to contribute toward the meeting and pursue many discussions of interest in parallel. The inspection comments are automatically placed in a structure that makes them easy to use for future reference.

However, we feel that eliminating FtF meetings is not desirable. Our studies show that not all the faults may be easily resolved in an asynchronous environment, and there may be a need for a synchronous meeting to resolve difficult faults, or to establish areas of common understanding. Moreover, asynchronous inspections make the participants' social interaction more difficult, losing some of the beneficial aspects of FtF work, such as team building[3].

We recommend that asynchronous inspection be used as a complement to synchronous inspection for co-located work groups. Asynchronous inspection should be performed first to resolve the majority of the faults, and provide valuable development history. Unresolved faults that remain should be inspected in a subsequent synchronous meeting.

Asynchronous inspection is especially useful when synchronous inspection is infeasible. An example of this would be inspection in geographically dispersed workgroups, for which the cost of synchronous inspection can be prohibitive.

Future Work

Our experience with distributed, asynchronous software inspection suggests a number of additional research problems:

Inspection of Artifacts from Different Perspectives:
Many methods of software development (e.g., the Unified Modeling Language [18]) involve looking at the software requirements or design from different (often graphical) perspectives. For instance, in developing object-oriented reactive systems, one is interested in both structural (e.g., the inheritance hierarchy) and behavioral views (e.g., finite state machines) of the system. An enhancement of distributed, asynchronous inspection would be to allow people to comment on an artifact in any view in which it appeared, and allow those comments to be seen by participants looking at other views of the same artifact, or at related artifacts (such as

parents of the artifact in an inheritance hierarchy).

Comparison Among Types of Inspection: We have reported that asynchronous inspection is feasible. Further, controlled studies would be required to determine the relative effectiveness (including cost-effectiveness) of FtF, distributed synchronous, and distributed asynchronous inspection where all three forms were feasible.

Distribution and Asynchrony Applied to Other Software Engineering Tasks: Our work thus far has concentrated on applications of distribution and asynchrony albstractions to software inspection. Further studies covering the application of these abstractions to a wide range of software engineering tasks are required to assess the feasibility of distributed, asynchronous software development.

ACKNOWLEDGEMENTS

We gratefully acknowledge the support of the National Science Foundation (grant numbers $NSF/IRI-9208546$ and $NSF/IRI-9410857$), ICEM Systems, Inc., and the research funds of the Graduate School of the University of Minnesota.

REFERENCES

[1] K. Ballman and L. Votta Organizational Congestion in Large-Scale Software Development *Third International Conference on the Software Process*, October, 1994.

[2] S. Bly, S. Harrison, and S. Irwin. Media spaces: Bringing people together in a video, audio, and computing environment. *Communications of ACM*, 36(1):28–47, Jan 1993.

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

[4] P. Dewan and R. Choudhary. Coupling the User Interfaces of a Multiuser Program. *ACM Transactions on Information Systems*, December 1994.

[5] P. Dewan and J. Riedl. Toward computer-supported concurrent software engineering. *IEEE Computer*, January 1993.

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

[7] M. Fagan. Design and code inspections to reduce errors in program development. *IBM System Journal*, 15(3):182–211, 1976.

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

[9] J. Gintell, J. Arnold, M. Houde, J. Kruszelnicki, R. McKenney, and G. Memmi. Scrutiny: A Collaborative Inspection and Review System. *Proceedings of the Fourth European Software Engineering Conference*, Garwisch-Partenkirchen, Germany, September, 1991.

[10] J. Gintell and G. Memmi. CIA: Collaborative Inspection Agent experience: Building a CSCW application for software engineering. *Workshop on CSCW Tools*, October 1992.

[11] O. Gotel and A. Finkelstein Contribution Structures. Second IEEE International Symposium on Requirements Engineering, March, 1995.

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

[13] P. Johnson, D. Tjahjono, D. Wan, and R. Brewer. Experiences with CSRS: An Instrumented Software Review Environment. *Proceedings of the Pacific Northwest Software Quality Conference*, October, 1993.

[14] V. Mashayekhi, J. Drake, W. T. Tsai, and J. Riedl. Distributed collaborative software inspection. *IEEE Software*, pages 66–75, September 1993.

[15] V. Mashayekhi, C. Feulner, and J. Riedl. CAIS: Collaborative Asynchronous Inspection of Software. *Second ACM SIGSOFT Symposium on the Foundations of Software Engineering*, December 1994.

[16] V. Mashayekhi, B. Glamm, and J. Riedl. AISA: Asynchronous Inspector of Software Artifacts. *University of Minnesota Technical Report TR-96-022*, March 1996. (Available from first author.)

[17] A. Porter, L. Votta, and V. Basilli. Comparing Detection Methods for Software Requirements Inspections: A Replicated Experiment. *IEEE Transactions on Software Engineering*, 21(6), June 1995.

[18] Rational Software Corporation Unified Modeling Language for Real-Time Systems Design. *http://www.rational.com/pst/tech_papers/uml_rt.html*, V0.91, September, 1996.

[19] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-Oriented Modeling and Design.* Prentice Hall, 1991.

[20] C. Thompson and J. Riedl. Collaborative Asynchronous Inspection of Software using Lotus Notes *University of Minnesota Technical Report TR-95-047*, June 1995. (Available from first author.)