User’s Name: John W Gintell

Document Title: scrutiny esec - RR

LaserWriter: Software Technology-2n1

Application: Microsoft Word

Date Spooled: Wednesday, January 12, 1994

Time Spooled: 9:25:28 AM

To appear in Proceedings of Fourth European Software Engineering Conference
Garmisch-Partenkirchen, Germany (Sept. 13-17, 1993)

Scrutiny: A Collaborative Inspection and Review System

John Gintell, John Arnold, Michael Houde, Jacek Kruszelnicki,
Roland McKenney, and Gérard Memmi

US Applied Research Laboratory
Bull HN Information Systems Inc.
300 Concord Road - MS 821A
Billerica, MA 01821 USA

(email: j.gintell@bull.com)

Abstract. This paper describes a Bull US Applied Research Laboratory project to build a collaborative inspection and review system called Scrutiny using ConversationBuilder from the University of Illinois at Urbana-Champaign. The project has several distinct aspects: technology oriented research, prototype building, experimentation, and tool deployment/technology transfer. Described are the design of the current operational version of Scrutiny for inspection-only, the evolutionary design of Scrutiny to handle various forms of review, and some initial thoughts on integration with other CASE frameworks and tools. The problem domain selected, the development environment, lessons learned thus far, some ideas from related work, and the problems anticipated are discussed here.

1 Introduction

Scrutiny is a collaborative system for inspection and review of software engineering work products. It is a distributed system that can be used by geographically separated users. Scrutiny uses general purpose mechanisms for integration with tools and CASE frameworks. It is tailorable for different software engineering process models. Scrutiny is intended for widespread use within our company to obtain usage experience with real users. As a starting point we implemented a working prototype restricted to inspection called CIA (Collaborative Inspection Agent) and obtained promising results from initial use. These results confirmed that our problem selection, technology choice, and initial design are sound. The results are guiding us in our next steps.

There have been a number of other efforts for computer assisted inspection and review described recently (see Section 7). Each describes a dimension of inspection that extends beyond the traditional view of what inspection is. Scrutiny differs from the work in these systems because it combines 1) a distributed collaborative environment, 2) integration with other tools using general purpose and standardized mechanisms, and 3) support for multiple process models.

This is a work-in-progress paper. Scrutiny has been underway for more than one year now[1]. We have a working prototype, some experimental users, favorable initial reaction from management as well as users and have obtained a first set of results. We continue to investigate, build prototypes and get practical experience

1 At the time of publication of this paper the project will have been underway for two years.

with some of the prototypes by making them robust and putting them into extensive practice. As a large company that is distributed in a number of locations worldwide, we have an ideal environment to test out some of these ideas in non-research oriented software engineering environments.

The remainder of this section describes the motivation for this project. Section 2 describes the CSCW technology selected as the basis on which to implement Scrutiny. Section 3 describes its design and implementation. Section 4 describes the current status and lessons learned. Sections 5 and 6 describe future work for Scrutiny. Section 7 describes related work in this problem domain. Section 8 contains our conclusions.

1.1 A Perspective on this Project

Inspection was chosen because it is a widely accepted practice within Bull [22] and it is well-defined. We wanted to build a tool that would be used by a variety of users who could give us usage-based feedback to guide future development. Selecting a widely accepted practice was a necessary condition for obtaining acceptance for the tool. Being a well-defined process gave us an excellent model to emulate as a starting point. Scrutiny was constrained to fit the Bull model so that using it would count as an "official" inspection.

Face-to-face inspection of all work products (e.g. documents, code, and tests) has proven to be an effective means for achieving higher product quality. Inspection is a well-defined process first developed by Fagan at IBM [6]. Inspection is usually performed without computerized assistance. It is generally done with a team of people who first prepare independently, and then meet in a room to perform a group inspection with individuals assigned to specific roles. It is a collaborative process because it takes teamwork to properly identify and classify the defects. After the inspection has been completed, defects identified during the meeting and other relevant information including metrics about the inspection itself are manually entered in various databases for later processing.

Uniformity of the inspection process is required so that results can be collected and rolled up for later analysis and process improvement. The motivation for this has been stimulated by the need to obtain ISO 9001 [9] certification and by the Software Engineering Institute sponsored Capability Maturity Model [8] [17].

There is a general desire to automate the inspection and review process, to allow it to be performed by geographically spread teams, and to integrate it with other environments (e.g. configuration management or bug tracking systems). Further, there is a continuum of process sub-models ranging from inspection to review-with-construction. This variety of forms of use led us to produce a tool that is configurable in many modes. We envision a number of advantages to be reaped with a distributed tool used for inspection and review and that is integrated with other tools and environments.

Computer-Supported Cooperative Work (CSCW) is currently a very active field. Many CSCW projects implement some form of shared writing and/or review. Scrutiny augments collaboration technology with controls at those points of people/application interaction governed by the rules of the inspection process.

2 Technology Selected - ConversationBuilder

We require a distributed system environment suitable for rapid application development. It must support collaborative work with simultaneous display of information on multiple users' screens under both application and individual user control. It must support user-role definitions so that the tools generated can comply with process rules and practices. It must be open and extremely flexible with respect to integration with other tools. It must be operational and supported. It should also be under active development so it can keep pace with new technologies and can evolve as requirements are generated by our applications and experiences.

ConversationBuilder (CB) [12] from University of Illinois at Urbana-Champaign was selected because of its architecture and the kind of problems it was designed to solve. CB supports a structured conversation model and was influenced by the Language - Action model by Winograd/Flores [23]; thus it is well suited to handling the discourse that occurs during inspection and review. CB's architecture is suitable for a wide variety of applications, for the linkage between them and for integration with other externally defined tools and environments. It satisfies all our requirements. It is under active development at the University of Illinois; Bull has become one of the sponsors of the work there.

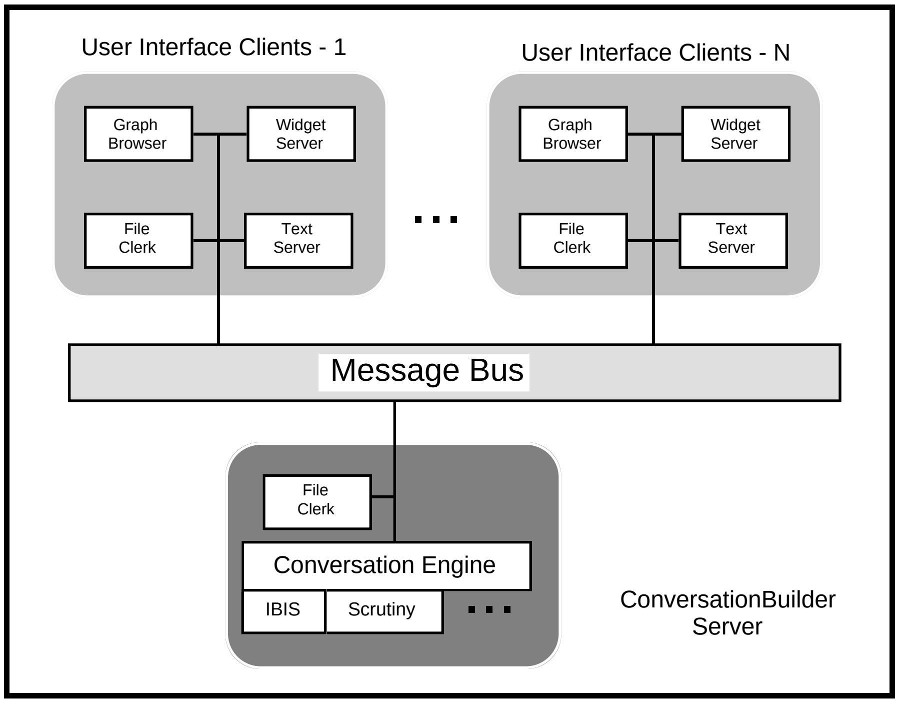


Fig. 1. ConversationBuilder Architecture

ConversationBuilder is a distributed system depicted in Figure 1; its major components are:

- the **Message Bus** - a multi-user control integration mechanism used to interconnect tool components; it acts as the vehicle for multi-cast message passing between all of the other components that comprise CB and its applications. The Message Bus is a FIELD-inspired form of integration [18].
- a **User Interface Suite** for each user that includes a **Motif-based Widget Server** to manage the user interface, a **Graph Browser**, a **Text Server**, and a **File Clerk** to manipulate files and their contents.

- the **Conversation Engine** itself whose job is to support collaboration and to manage the actual application activities. The applications are called protocols. A protocol is a CLOS (Common Lisp Object System) description of a conversation flow that is enacted by the engine.

CB is written in CLOS and C and runs on UNIX™; the applications themselves are also written in CLOS enabling them to inherit the classes defined in CB and thus build upon its mechanisms. In addition to Scrutiny, there are CB protocols either in experimental or operational form for a wide variety of collaborative processes: IBIS[2] [3], Design Rationale, Shared Whiteboard, Shared Editor, WorkFlow, Configuration Management, and Bug Tracking.

3 Description of Scrutiny

3.1 Inspection Roles as implemented by Scrutiny

Based on the Fagan model, the inspection roles are as follows:

- a **Moderator** to control the inspection activities from beginning to end. He or she chooses the inspection team, coordinates the entire process, moderates the group meeting, and ensures that all steps are taken correctly.
- the **Inspectors** to do the work of defect finding. Their work is performed first privately to do preparation and then as part of the group to identify and classify the defects. The Moderator also acts as an Inspector.
- a **Reader** to read (or paraphrase) every line/section to all participants during the inspection so that the entire work product is covered.
- a **Recorder** to classify and record defects. All records of the inspection are made by the Recorder.
- a **Producer** of the work product. The Producer is present to supply additional information about the work product. A second but less important function is to collect non-defect comments gathered during the inspection that would be useful for future revisions to the work product.

Scrutiny does not implement all these roles explicitly. The Inspector and Moderator roles are kept intact. The Producer is not distinguished from other Inspectors. The Moderator performs the Reader role by highlighting the current focus of attention in the work product on each user's screen. The Recorder role is performed by the Moderator in collaboration with the Inspectors and the application itself.

3.2 Inspection Stages in Scrutiny

The inspection has four sequential stages: **Initiation, Preparation, Resolution, and Completion**.

- **Initiation** for the Moderator to set up the inspection by obtaining the work product(s) to be inspected, identify and invite the participants, and do all of the up-front work needed to ensure a successful inspection.
- **Preparation** for the Inspectors to examine the work product and prepare comments and questions (which we call annotations). In this stage each of the Inspectors work independently.

2 IBIS is an acronym for Issue Based Information System, a structured method for discourse on issues; it is considered a classical CSCW application.

- **Resolution** for the formal defect processing. The Moderator paces the meeting by stepping through the work product. All Inspectors contribute collectively by reading the results of others' preparation and then creating more annotations (defects, questions, actions for the future, ...). The Recorder classifies the defects based upon the annotations.
- **Completion** for the results of the Inspection to be processed and prepared for transmission to other stages of the software development process. For example, at a later time, the Producer would remove the defects by making a revision of the work product (perhaps requiring a reinspection). This stage is performed by the Moderator and Recorder.

3.3 Artifacts created by Scrutiny

Items entered by Inspectors are called annotations and are initially labeled as a question, potential defect, remark, or reply. They later may be modified, made obsolete or relabeled. Each defect is further classified; they are treated as a separate class of artifacts because of their importance to the process. There is a polling procedure invoked by the Moderator to obtain consensus; the results are preserved as artifacts. At the end of the inspection a variety of reports may be prepared to document the results and actions taken during the inspection. The defects are also formatted in a manner suitable for insertion into defect managing systems.

Figure 2 describes the attributes of the stages of Scrutiny to show the relationship among the artifacts, roles and stages.

	INITIATION	PREPARATION	RESOLUTION	COMPLETION
PARTICIPANTS	• Moderator • Producer	• Inspectors	• Moderator • Inspectors • Producer • Recorder	• Moderator • Recorder
SYNCHRONY		• asynchronous	• synchronous	
ACTIONS	• get work product • select participants • schedule meeting	• study materials • find related info • make annotations	• refine annotations • identify defects • classify defects • resolve issues	• finish defect classification • prepare reports • close meeting
ARTIFACTS CREATED		• annotations	• annotations • defects • polls	• defect list • reports • statistics

Fig. 2. Scrutiny Stages and their Attributes

3.4 Scrutiny User Interface

The Scrutiny user interface attempts to balance two different points of view on collaborative inspection and review: the process and role points of view. The user interface may be affected by either of these points of view. The effect is realized by enabling or disabling menu selections and/or buttons in the appropriate windows.

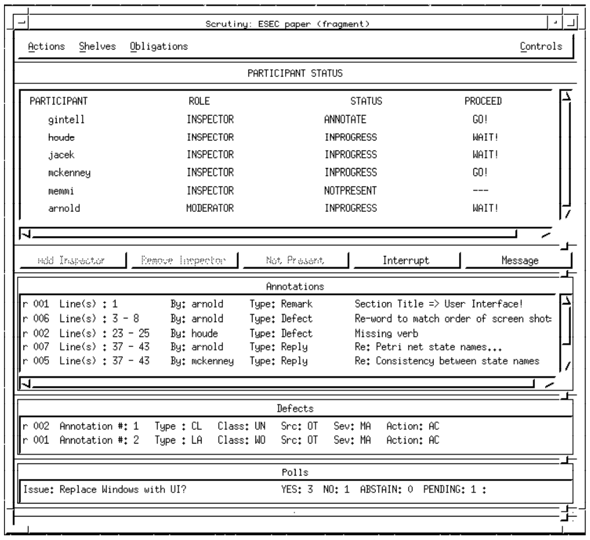
The process orientation affects the user interface depending on the current state of the inspection/review process. The process, described with Petri nets (see Section 3.5), defines the states a role can reach at a given point in the inspection. This results in a somewhat modal interface, but the modality is necessary to accurately reflect the inspection model we have chosen. For example, an Inspector cannot create a new annotation when a poll is in process because the defined process does not allow it.

The role orientation affects the user interface depending on the role of the individual user. For example, an Inspector can neither convene the meeting (i.e., move it from Preparation stage to Resolution stage) nor change the current focus of the work product; these actions are restricted to Moderator and Reader roles.

Two types of information can be displayed in the windows: private and shared. Private information is seen only by the user who created it. Shared information can be accessed by any participant in the inspection/review. The shared information can be presented in a synchronous or asynchronous manner. Synchronous, shared information is managed such that all changes are immediately reflected in each user's view. Asynchronous, shared information is managed by presenting the information when requested by the user. For instance, the annotations list in the middle of Figure 3 is updated (during Resolution stage) with a new entry whenever any user creates a new annotation; thus, it is asynchronous, shared information. The text of the annotation is shared, synchronous information, however, since it is only presented to the user when he/she acts to display the full annotation.

The Scrutiny Control Window (Figure 3) acts as a control panel for the application. Each participant sees a copy of this window. The Participant Status information at the top is a shared, synchronous pane; each user sees exactly what the other users see. As a user acts, his or her status is updated. For instance, Figure 3 shows user 'gintell' is annotating. The buttons are activated or deactivated depending on their applicability to the particular role and state of the inspection.

The annotation pane contains a list of annotations on the work product that have been made so far. This list can be sorted in a number of ways. The defect pane contains a list of defect reports that have been classified. Finally, the poll pane lists the polls that have been taken during this inspection and their results. Selecting an annotation, defect, or poll will open a window for the object showing it in more detail. The annotation, defect, and poll panes are shared, synchronous panes. As a new object of the given type is created, a new entry appears in the appropriate



PARTICIPANT	ROLE	STATUS	PROCEED
gintell	INSPECTOR	ANNOTATE	GO!
houde	INSPECTOR	INPROGRESS	WAIT!
jacek	INSPECTOR	INPROGRESS	WAIT!
mckenney	INSPECTOR	INPROGRESS	GO!
memmi	INSPECTOR	NOTPRESENT	---
arnold	MODERATOR	INPROGRESS	WAIT!

```
r 001  Line(s) : 1        By: arnold    Type: Remark     Section Title => User Interface!
r 006  Line(s) : 3 - 8    By: arnold    Type: Defect     Re-word to match order of screen shot
r 002  Line(s) : 23 - 25  By: houde     Type: Defect     Missing verb
r 007  Line(s) : 37 - 43  By: arnold    Type: Reply      Re: Petri net state names...
r 005  Line(s) : 37 - 43  By: mckenney  Type: Reply      Re: Consistency between state names
```

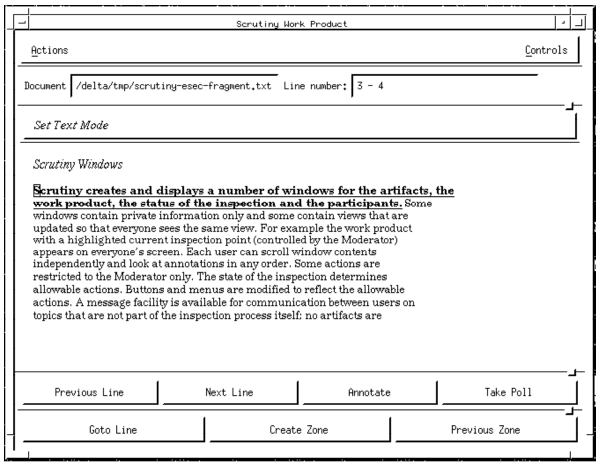
```
r 002  Annotation #: 1  Type : CL  Class: UN  Src: OT  Sev: MA  Action: AC
r 001  Annotation #: 2  Type : LA  Class: WO  Src: OT  Sev: MA  Action: AC
```

```
Issue: Replace Windows with UI?             YES: 3  NO: 1  ABSTAIN: 0  PENDING: 1 :
```

Fig. 3. Scrutiny Control Window

pane. However, accessing the new object is asynchronous since the object itself is only displayed when the user requests it. This allows the users to look at annotations in any order (just as they can leaf through the work product in a face-to-face inspection).

The Scrutiny Work Product Window (Figure 4) presents a shared view of the Work Product being inspected. The top pane shows the pathname of the work product and the line(s) of the current focus in the document. The primary pane displays the work product itself. The bold and underlined region is the current focus. Each user can scroll this pane independently though they are not allowed to modify it. Beneath this, there are two rows of buttons. The navigation buttons (Previous Line, Next Line, Go To Line, and Create Zone) and the Take Poll button are active for the Moderator only. The Annotate button is used to create an annotation about the current focus.



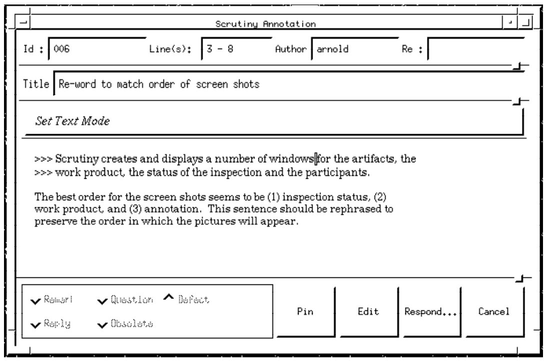
Document /delta/tmp/scrutiny-esec-fragment.txt Line number: 3 - 4

Scrutiny Windows

Scrutiny creates and displays a number of windows for the artifacts, the work product, the status of the inspection and the participants. Some windows contain private information only and some contain views that are updated so that everyone sees the same view. For example the work product with a highlighted current inspection point (controlled by the Moderator) appears on everyone's screen. Each user can scroll window contents independently and look at annotations in any order. Some actions are restricted to the Moderator only. The state of the inspection determines allowable actions. Buttons and menus are modified to reflect the allowable actions. A message facility is available for communication between users on topics that are not part of the inspection process itself; no artifacts are

Fig. 4. Scrutiny Work Product Window

The Scrutiny Annotation Window (Figure 5) is used to create or display the details of an annotation. The top panes include information about the annotation such as the focus on which this annotation was made, author, and title. The middle pane is a buffer in which the body of the annotation is typed or displayed. At the bottom, there are radio buttons which categorize the annotation and another set of buttons for frequently used actions on the annotation.



Id : 006 Line(s): 3 - 8 Author arnold Re :

Title Re-word to match order of screen shots

>>> Scrutiny creates and displays a number of windows for the artifacts, the
>>> work product, the status of the inspection and the participants.

The best order for the screen shots seems to be (1) inspection status, (2) work product, and (3) annotation. This sentence should be rephrased to preserve the order in which the pictures will appear.

Fig. 5. Scrutiny Annotation Window

3.5 Model of the Resolution stage

The Resolution stage of Scrutiny models face-to-face inspections as closely as possible. Thus the participants are engaged in synchronized collaboration to adhere to the requirements of the inspection process. It is this property of inspection that distinguishes Scrutiny from many of the other cooperative applications where the cooperation is controlled by the users and not the application.

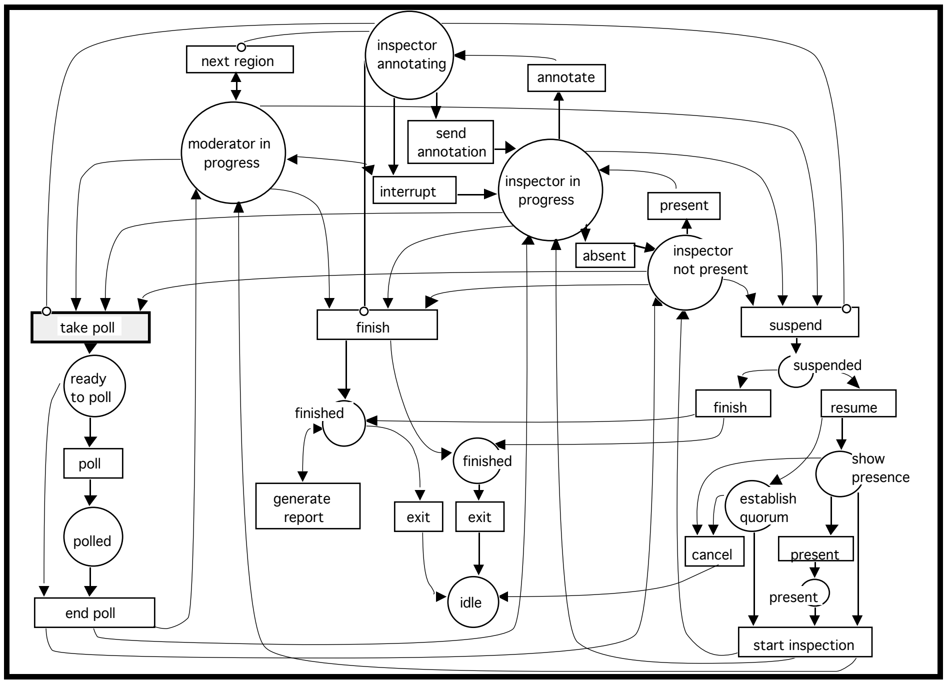


Fig. 6. High level Petri net Diagram for Scrutiny Resolution Stage

This high level Petri net (Figure 6) describes a simplified version of the behavior of the Resolution/Completion stages of Scrutiny. Each place (circle) represents a possible state for a Moderator or an Inspector. Each transition (box) represents a possible action. The intent is to provide an idea of the level of interaction between the two roles during these stages, not to describe a precise design of this stage[3].

As an example, consider the poll sub-process. The *take poll* transition (highlighted in Figure 6) is enabled when no Inspector is annotating and the Moderator is *in progress*. Then, every Inspector (in the state *in progress* or *not present*) and the Moderator all move into the state *ready to poll*. Each respondent may vote or not; however, once having voted the vote may not be changed. The Moderator decides when to enable the transition *end poll*. Whether they responded to the poll or not, each participant then returns to the place occupied prior to the taking of the poll.

Note also the *interrupt* transition, which was not implemented in the early versions of Scrutiny. This transition was added to permit the Moderator to force Inspectors out of *inspector annotating* and into *inspector in progress*. Otherwise, the Moderator was unable to move to the *next region*, *take poll* or even *suspend meeting* if an Inspector should walk away from his or her workstation while in *inspector annotating*.

3 The full design requires labeling edges and transitions, giving an interpretation of the entire diagram, and to complete the set of individual actions allowed (most of which do not change the global state of Scrutiny).

4 Current Status and Lessons Learned

At the time of submission of this paper, Scrutiny is used by the Scrutiny team and another project's team to use it to perform inspections and find and classify defects[4]. This paper, some code and a number of other documents have been inspected with Scrutiny. We have been performing demos and working with several other groups within Bull for informal training and evaluation. Scrutiny (named CIA at the time) was demonstrated at the ACM 1992 Conference on Computer-Supported Cooperative Work in Toronto, Canada (November 1992) and at the ACM SIGSOFT 1992 Fifth Symposium on Software Development Environments in McLean, Virginia (December 1992).

We are increasing the volume of experimentation with volunteers from other development groups and will learn more during the next few months. We have already made a number of improvements suggested by these initial users and have a number of ideas for developing a much improved system. The remainder of this section summarizes what we have learned and done.

4.1 Effectiveness of Computer-Supported Inspection

Inspections done with Scrutiny are at least as effective as those performed via face-to-face meetings; they appear to take about the same amount of time and identify defects as effectively as face-to-face inspections. We found that inspection-trained users could start using Scrutiny effectively with less than 2 hours of training, and in several cases they were able to learn "on-the-fly".

Some significant advantages over the face-to-face process have been demonstrated. Because the defects are collected in machine-readable form for subsequent processing by other tools, clerical and administrative work that is performed after an inspection is eliminated. This was one of the first features we added in response to user requests. Because all of the non-defect comments are captured, the Producer doesn't have to waste valuable inspection time recording them or collecting Inspectors' marked up papers or listings. Similarly, the Recorder doesn't have to pause the inspection while doing bookkeeping or manual transcription during the inspection meeting. We have some evidence that Inspectors do a more comprehensive job of preparing because the results of preparation will be seen by others; this might result in more defects being found.

4.2 Stages and Roles

The initial version of Scrutiny did not have the Preparation stage integrated in the tool. Based upon initial experience, this was the highest priority improvement to make to Scrutiny and it is now operational. This allows Inspectors to work independently while doing preparation with the tool and then enables reuse of annotations made during the Preparation stage during the Resolution stage; this reuse enables a noticeable speedup of the Resolution stage.

Our original assumption was that the computer assistance was sufficiently powerful that the Recorder role could be handled by the Moderator. We have discovered that the Recorder role must be explicitly implemented - the inspection is slowed down if the Moderator must always do all of the Recorder's work as well as controlling the meeting and inspecting.

4 The most exciting times have been while using an imperfect version of Scrutiny to inspect design documents about Scrutiny itself since the annotations and comments are all self-referential in real-time.

4.3 Performance

One of our initial assumptions about performance was that the Message Bus would be a potential bottleneck. We obtained considerable data about the performance of the underlying mechanisms (e.g., the Message Bus and the Conversation Engine) while using Scrutiny [21]. The preliminary data showed that the bottleneck was in the Conversation Engine. As a result of this data the CB team redesigned the underlying mechanisms and with some rework on our part we obtained a 3-5 times improvement. We also did some experimentation with CB components distributed over a WAN and found that if the network was at least 50KB/sec Scrutiny performance was essentially the same as our day-to-day use on a LAN. There is considerably more work to do in collecting and analyzing data, but these first experiments done at this early stage of the project have been valuable to us.

4.4 User Interface

As a result of the experimental usage we have made a number of modifications to the number and format of windows to simplify the user interface. We replaced some pull-down menus with buttons for frequently used requests. For example, we added a *respond* button to the annotation window since responding was a frequent use for annotations. We modified Scrutiny to indicate what portions of the work product were already inspected. We added support for the use of the CB graph browser facility to give users an additional means for looking at annotations.

The experience strongly supported our expectations about the influence the user interface will have on the overall usability and acceptance of Scrutiny; a few minor changes have already made a big difference; however, substantial improvement remains to be made.

4.5 ASCII-only Work Products

Work products processed by Scrutiny are restricted to ASCII files only. Documents that are produced by most word processors can have text-only versions created, but diagrams, charts and formatting can not be inspected. Scrutiny has been used to inspect documents where the diagrams (if present) were inspected manually. In the future we will address this through integration with other tools that can process complex documents.

Inspection is usually performed by moving the focus on the work product from beginning to end in sequential order. How to handle complicated diagrams or charts is a problem that requires further study.

For code inspection, ASCII text format is usually sufficient. A different issue arises when the structure of the code itself suggests other possible orderings for the inspection. Integration with code analysis tools will allow “structured” inspections to be performed; this should prove to be an additional advantage of computer-assisted inspections over face-to-face inspections.

4.6 Audio Channel

Face-to-face meetings have the advantage of high bandwidth communication using voice, gestures, and informal and spontaneous hand-written notes or drawings. Such communication has the disadvantage of usually not being preserved. Scrutiny was designed so that it can work with no audio facilities. We took this approach so that the absence of audio support would not preclude use of Scrutiny. In general, such facilities are not readily available particularly between geographically separated locations. Usage thus far shows that effective inspections can be done without audio.

We have done some experimentation using Scrutiny in the same room with people talking, with teleconference, and with audio facilities built into the work stations to see the effect on the inspection. Synchronization and quick question/answering is much easier with audio and speeds up the Resolution stage. As a result of this experimentation we are adding some more easy-to-invoke discussion facilities than that offered by the creation of annotations to reap some of these advantages of audio. Handling mixed capabilities is an open research issue.

5 Generalizing the process

The work to evolve Scrutiny for activities similar to inspection such as reviews and requirements collection is well under way. In essence, this changes the rules concerned with operations performed by the various roles and during the several stages to make Scrutiny's behavior less constrained than required for inspection.

In inspection, the Preparation stage has each Inspector working independently whereas the Resolution stage has all work synchronized by the Moderator. For review, neither independent preparation activity nor synchronized resolution activity is required by the process. Uncontrolled shared preparation can be useful for people to work in an interactive fashion while elaborating on their incompletely formed initial thoughts. On the other hand, moderated inspection-like work is useful to assist in reaching closure on issues. To address this in Scrutiny, we will divide the Preparation stage into two sections where the first is for private work and the second is for collaborative shared preparation. The Resolution stage remains the same although with effective shared preparation it may be very short. Classified defects are not produced during review and thus the artifacts produced by the Resolution stage are different.

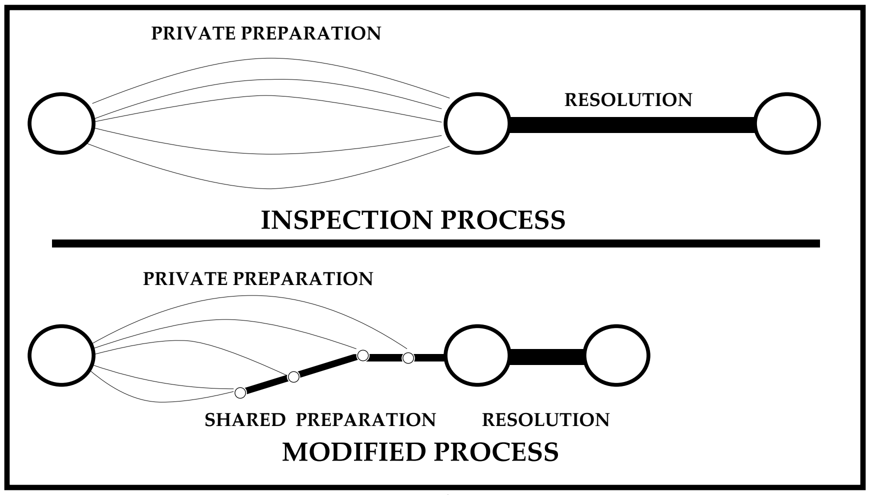


Fig. 7. Modified process

Figure 7 shows a comparison between the inspection process and the process modified for review. The thin lines represent private preparation activities by participants, thick lines represent collaborative preparation, and the very thick lines represent synchronized resolution activities. Large circles represent stage changes and small circles represent the joining of participants for shared preparation. In this process instantiation, each participant switches from private to shared participation at his or her own selected time. As each participant joins this shared participation he or she sees the previously joined participants' annotations and vice-versa. This permits collaboration to begin for some people while others continue private participation, joining later. Scrutiny will permit a variety of policies as to when and how this switching occurs for different review models. This flexibility will also enable experimentation with changes to the Inspection process.

6 Future Work for Integration with Tools and Environments

Inspection and review should be more than isolated processes acting on a fixed set of work products that are "input" to the activity. Scrutiny needs to be integrated with other tools. Some examples include a static analyzer to use in code inspections, the tool used to deposit data in a metrics database, and a document processor for non-ASCII text so compound documents can be managed and displayed by the producing software instead of having to be translated to another form for processing by Scrutiny.

There are a number of significant enhancements that we can make by using data integration, control integration, process integration, and distributed processing services to integrate CB/Scrutiny with other tools, services, and information. In the remainder of this section we briefly describe this aspect of our research. Much of this work is exploratory with unresolved issues that will take considerable time to resolve.

6.1 Data Integration for Tool Access

The introduction of an external OMS (object management system) presents a challenge because CB has its own interim persistent object store. Our initial work with data integration uses the GIE Emeraude implementation [7] of the Portable Common Tools Environment (PCTE) [5]. In an environment where all of the software engineering work products are maintained by an OMS, Scrutiny access to this OMS will enable two significant enhancements to Scrutiny:

- During an inspection, identified defects are associated with regions of the work product. After the inspection, the removal of these defects is managed as part of the standard software engineering process; but not all may be removed at the same time. Traditionally, the defects are managed in a separate defect tracking system. With the use of an OMS that includes versioning of objects, the defect reports can be explicitly associated with the objects themselves.
- Scrutiny should be able to easily find and display up-to-date versions when needed. While inspecting a pre-defined set of work products, it is often desired to examine a related set of work products whose need was not necessarily known at the start of the inspection (e.g., previous versions when inspecting code, or documents in the reference list of a design document).

6.2 Control Integration for Tool Access

The Message Bus of ConversationBuilder is Scrutiny's control integration mechanism. It allows tools to send commands and notifications to the other tools that are connected to the Message Bus. It is influenced by FIELD as shown in its multicast mechanism that connects tools *through* the bus rather than directly to each other. Unlike many other FIELD-inspired messaging systems, the Message

Bus is a multi-user system; this enables messages sent due to one user's actions to be "seen" by other Scrutiny users. We see further use of this aspect of the Message Bus to control each user's interaction with non-Scrutiny tools as well as for the control actions of the Scrutiny protocols. FIELD-like systems have also demonstrated considerable value for the loose integration of existing UNIX tools [1]. Thus the Message Bus is well suited as the basis for interapplication communication between Scrutiny applications and encapsulation of existing tools such as the static analyzer mentioned earlier.

6.3 Process Integration

We are investigating various process modeling and work flow tools. One such tool is the Marvel system, a rule-based software engineering environment developed at Columbia University [11]. Within this system it is possible to encode the rules that describe the Software Development Process and use these rules to control the execution of the tools that perform the process steps. We are investigating integration of Marvel and CB and are experimenting with Scrutiny in this merged environment.

6.4 Distribution and Connection

We want to use Scrutiny for inspections and reviews where not all of the participants are at the same site and in fact may be separated by many network hops and time zones (Figure 8). To support this, we have been studying Scrutiny/CB/Message Bus communication to understand the traffic pattern, simulate various modes of operation, and prepare for revised distribution architectures.

Another issue in this domain is that Scrutiny and CB itself currently assume that all participants must be connected to use the system. CB is sufficiently robust that an unconnected user doesn't prevent the others from continuing and he or she can later reconnect and catch up. The advent of notebook computers and other detachable systems yields the possibility of some work being done on a non-networked system. In fully connected Scrutiny, even though the individual users are not aware of each other's preparation, the system is. The challenge here is to correctly synchronize this independent work with that done by the other participants at a later time.

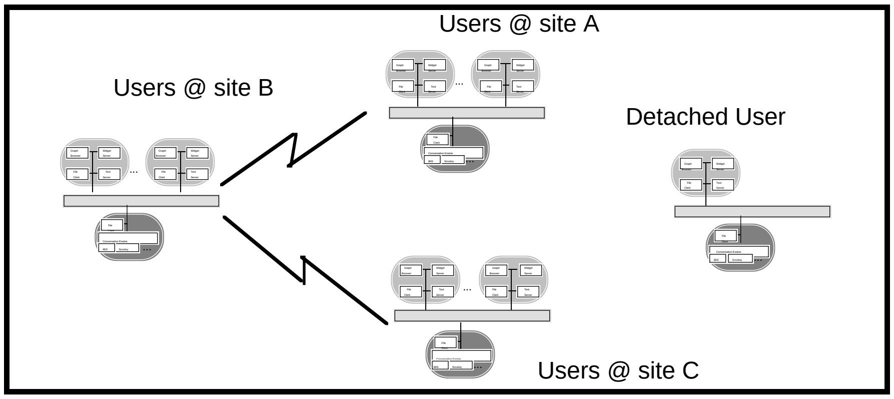


Fig. 8. Scrutiny Distributed and Detached

7 Related Work on Inspection

This section describes some key points we have observed by study of other people's work. We have loosely classified them into ideas that improve comprehensiveness, ideas that decrease the time and effort, and ideas that are related to process change.

7.1 Inspection Comprehensiveness

One of the problems associated with inspections is that it is difficult to even find most of the defects. Various techniques are used with the most common technique involving building checklists and having all inspectors use these checklists. InspeQ by Knight/Myers [13] has the capability of displaying checklists and corresponding standards with cross referencing and examples in specific displays and to guide the Inspectors in their work. This illustrates another advantage of computer assisted inspection by making this form of information readily available to the inspectors and also ensuring that up-to-date information is available.

Martin/Tsai [14] discuss a technique where multiple teams inspect the same requirements document to maximize the likelihood of finding defects. Then a common moderator collates the results to produce a single defects list. They have applied this to mission-critical software, with excellent results. Of course, one can't afford to inspect every item more than once. Applying this technique to the requirements phase is exactly the right place since undetected and thus remaining defects in requirements documents have the most expensive consequences. This technique lends itself to computer assisted inspection in managing and coordinating these inspections, particularly if they are geographically distributed where traditional means of coordination would be slow. This is an example of where a change to improve inspection comprehensiveness introduces a major change to the model of inspection which has significant implications to the design of the system.

As described earlier, there is a desire to give inspectors/reviewers access to related materials that were not identified at the start of the inspection. CSRS by Johnson/Tjahjono [10] implemented a scheme for managing and viewing other related artifacts during the inspection and have obtained good results during reviews.

7.2 Time and effort needed to perform Inspections

Inspecting large work products cannot be done in a single session because people cease to be effective after inspecting for a period of time. The traditional way to solve this problem is to do the first 200 lines in the first session, the second 200 lines in the second, etc. InspeQ (Knight/Myers) describes a Phased Inspection where the division is logical instead of physical and each division is called a Phase. Inspectors are assigned specific topics to cover with some phases being assigned to one person only (e.g. looking for spelling errors) while other phases are done collaboratively. InspeQ itself assists the Moderator in managing the process. It can easily be seen that this allows more parallelism and can consume less total effort than the traditional division. (Imagine going to the seventh 2-hour meeting and trying to remember the context that was set in the first meeting 2 weeks ago.) This is a good example of a major change in the inspection model that is more manageable with an automated tool.

Russell [19] describes results, costs, and cost savings in a very large Bell/Northern Research project performed in 1988. He shows how large the inspection effort is and provides additional motivation for computer assistance to manage it and make inspection more efficient.

7.3 Other Changes to the Inspection Process

Johnson/Tjahjono, in describing CSRS, state that a system allows one to collect considerable empirical data about the inspection process and use that data to test various assumptions for validity. For example, they point out that the effectiveness of the use of checklists can be questioned. Inspection results with and without checklist usage can be compared in a rigorous fashion more readily using a system like CSRS than by ad-hoc or anecdotal methods.

Icicle [2] is an inspection system built at Bellcore to support users who are in the same room. Like Scrutiny, it was done to explore a number of research issues, gain payoff from use, extend it to distributed inspections and reviews, and to try integration into other frameworks. The authors felt that such work would enable process changes to produce higher quality software more cost effectively. Similar issues are presented in [4].

In a separate experiment in our lab [15], a prototype inspection tool modeled after Scrutiny was built in EAST [20], an environment based upon PCTE. In this tool, the work product being inspected is stored in the PCTE Object Management System. When defects are found, they are stored as hyper-links in versions of the work product. This allows the defects to be carried along with the object itself. We will apply the results of this work to Scrutiny.

8 Conclusion

This paper presents an overview of the Scrutiny system, the architecture and description of its main features, and to some extent a view on its future evolution. This presentation should be understood as an example of the benefit that Software Engineering tools can draw from CSCW technology. We deliberately chose to focus on inspection in the software life-cycle both because this process is particularly collaborative and because, to our knowledge, no “off-the-shelf” efficient tools for inspection exist today.

The first experiences have been very encouraging and enriching. In spite of some shortcomings in the current implementation, some individuals prefer inspecting with Scrutiny, citing the advantages of having the results in electronic form. While enhancing Scrutiny, we will continue experimentation by product development teams. Developing iteratively and working with users will guide us in the next improvements. After new experimentation with Scrutiny we expect to be able to share more results in the near future.

The goals of this project include both building an inspection tool and testing the integration technologies. Our relationship with the ConversationBuilder development team lets our experience influence CB evolution and lets us rapidly take advantage of the subsequent improvements.

Finally, we would like to stress the importance of integration with audio, video, and other multimedia facilities. Widespread use of such technology is currently precluded by cost and availability. However, we have no doubt that the appeal of multimedia is extremely strong.

9 Acknowledgments

This paper could not have been written without the work of many people who have worked with us on the design or have been experimental users. Simon Kaplan's ConversationBuilder team and in particular, Doug Bogia and Bill Tolone have been invaluable.

10 References

1. John E. Arnold and Gérard Memmi. Control Integration and its Role in Software Integration. In: Toulouse '92 Fifth International Conference: Software Engineering & its Applications, Proceedings (December 1992)

2. L. Brothers, V. Sembugamoorthy, and M. Muller. ICICLE: Groupware for Code Inspection. In: Proceedings of CSCW '90, ACM Press (October 1990)

3. J. Conklin and M. Begeman. gIBIS: A hypertext tool for exploratory policy discussion. In: Proceedings of CSCW '88 (1988)

4. Janet Drake, Vahid Mashaykhi, John Riedl, and Wei-Tek Tsai. Support for Collaborative Software Inspection in a Distributed Environment: Design, Implementation, and Pilot Study. University of Minnesota Technical Report, TR 92-33 (June 1992)

5. ECMA. Portable Common Tool Environment (PCTE) Abstract Specification. ECMA-149 (December 1990)

6. Michael E. Fagan. Design and Code Inspections to Reduce Errors in Program Development. In: IBM Systems Journal, Vol. 15 - No 3 (1976)

7. The Emeraude Environment, Reference Manual Set, GIE Emeraude (July 1990)

8. W.S. Humphrey. Managing the Software Process. Addison-Wesley, Reading, MA, (1989)

9. Quality Systems - Model for Quality assurance in design/development, production, installation and servicing: ISO 9001. International Organization for Standardization (1987)

10. Philip Johnson and Danu Tjahjono. Improving Software Quality through Computer Supported Collaborative Review. University of Hawaii, ICS-TR 92-04 (1992)

11. Gail E. Kaiser, Peter H. Feiler, and Steven S. Popovich. Intelligent assistance for software development and maintenance. In: IEEE Software, 5(3) - 40-49 (May 1988)

12. Simon M. Kaplan, William J. Tolone, Douglas P. Bogia, and Celsina Bignoli. Flexible, active support for collaborative work with ConversationBuilder. In: Proceedings of CSCW '92, ACM Press (November 1992)

13. John C. Knight and E. Ann Myers. Phased Inspections and their Implementation. In: ACM - Software Engineering Notes, Vol. 16 - No 3, ACM Press (July 1991)

14. Johnny Martin and W.T. Tsai. N-Fold Inspections: A Requirements Analysis Technique. In: Communications of the ACM, Vol. 33 - Number 2, (February 1990)

15. Reza Morakabati. PCTE-based Inspection Tool - Design and Implementation. Bull USARL Research Report, RAD/USARL/93018 (1993)

16. Susanna Opper, Henry Fersko-Weiss. Technology for Teams. Van Nostrand Reinhold (1992)

17. Mark C. Paulk, et.al. Capability Maturity Model for Software. CMU/SEI-91-TR-24, Software Engineering Institute (August 1991)

18. Steve Reiss. Interacting with the FIELD Environment. Brown University Department of Computer Science, Technical Report No CS-89-51 (May 1989)

19. Glen W. Russell. Experience with Inspection in Ultralarge-Scale Developments. In: IEEE Software (January 1991)

20. EAST Environment, Manual set. SFGL (1992)

21. Li-Tao Shen, Pascal Petit, and Patrick Denimal. Performance Evaluation of the Message Bus of ConversationBuilder through the Scrutiny application. Bull USARL Research Report. RAD/USARL/93019 (1993)

22. Edward F. Weller. Lessons Learned from Two Years of Inspection Data. In: Proceedings of The 3rd Annual Applications of Software Management Conference (November 1992)

23. Terry Winograd and Fernando Flores. Understanding Computers and Cognition. Addison-Wesley (1987)