An Instrumented Approach to Improving Software Quality through Formal Technical Review

Research Paper

Philip M. Johnson
Department of Information and Computer Sciences
University of Hawaii
Honolulu, HI 96822

Abstract

Formal technical review (FTR) is an essential component of all software quality assessment, assurance, and improvement techniques. However, current FTR practice leads to significant expense, clerical overhead, group process obstacles, and research methodology problems.

CSRS is an instrumented, computer-supported cooperative work environment for formal technical review. CSRS addresses problems in the practice of FTR by providing computer support for both the process and products of FTR. CSRS also addresses problems in research on FTR through instrumentation supporting fine-grained, high quality data collection and analysis. This paper describes CSRS, a computer-mediated review method called FTArm, and selected findings from their use to explore issues in formal technical review.

1 Introduction

Assessment and improvement of software quality is increasingly recognized as a fundamental problem, if not *the* fundamental problem confronting software engineering in the 1990's. Low quality has always figured prominently in explanations for software mishaps, from the Mariner I probe destruction in 1962, to AT&T's 4EES switching circuit failure in 1992. More recently, however, low software quality has also been implicated in competitive failure on a corporate scale [2], as well as in loss of life on a human scale [18].

Research on tools and techniques to improve software quality shows that formal technical review (FTR) provides unique and important benefits. Some studies provide evidence that FTR can be more effective at discovering errors than testing, while others indicate that it can discover different classes of errors than testing [20, 4]. In concert with other process improvements, Fujitsu found FTR to be so effective at discovering errors that they dropped system testing from their software development procedure [2]. FTR forms an essential part of methods and models for very high quality software, such as Cleanroom Software Engineering and the SEI Capability Maturity Model. Finally, FTR displays a unique ability to improve the quality of the producer as well as the quality of the product.

FTR always involves the bringing together of a group of technical personnel to analyze an artifact of the software development process, typically with the goal of discovering errors or anomolies, and always results in a structured document specifying the outcome of review. Beyond this general similarity, specific approaches to FTR exhibit wide variations in process and products, from Fagan Code Inspection [9], to Active Design Reviews [22], to Phased Inspections [17].

Despite its importance and potential, the state of both FTR practice and research suffers from problems that hinder its adoption and effective use within organizations. First, most FTR methods are manual, prone to breakdown, and highly labor-intensive, consuming a great deal of expensive human technical resources. For example, a recent study documents that a single code inspection of a 20 KLOC software system consumes one person-year of effort by skilled technical staff [24]. Second, high-quality empirical data about the process and products of FTR is difficult to obtain and comparatively evaluate. Only Fagan code inspection enjoys a relatively broad range of published data about its use and effectiveness. The lack of such research data makes it difficult to compare different methods, improve the process, or match a method to a particular organizational culture and application domain.

For the past two years, we have been experimenting with a computer-supported cooperative work environment called CSRS (for Collaborative Software Review System), coupled with a method called FTArm (for Formal, Tech-

nical, Asynchronous review method). This system and method are designed to address problems in both the practice of and research on FTR. Laboratory studies demonstrate that a highly instrumented, collaborative environment that puts most processes and products of FTR on-line can lead to increased user satisfaction, enhanced capture of significant knowledge about software, useful new measures of FTR processes and products, and finally, higher quality software.

The primary goal of this paper is to inform the software engineering research and development community on how computer-mediated FTR can not only address certain problems associated with manual approaches, but can also provide high quality and low cost data useful for improving the process and products of FTR. We believe our experiences with CSRS and FTArm provide useful insights to the designers of current and future formal technical review systems.

1.1 Issues in FTR research

Research on FTR has led to a wide variety of review methods. However, the current state of FTR research fails to provide clear guidance to an organization in choosing a well-suited review method.

One deficiency in the literature is the lack of high quality, empirical studies comparing different review methods to each other. Past studies compare review to testing [3, 20] or compare different factors within a single review method (usually Fagan's inspection), such as the effect of the number of participants or group composition on review effectiveness [5, 19]. In addition, although these latter comparative studies claim to use the same approach (Fagan code inspection), ambiguities and inconsistencies in the description of the review method indicate that this key factor was not controlled sufficiently to allow cross-study comparison.

Another problem with the current state of research on review is conflicting and/or anecdotal explanations of the causal factors underlying review outcomes. For example, researchers have variously attributed an FTR method's effectiveness to general group interaction [8, 23], producer–reviewer interaction [20, 22], lack of producer–reviewer interaction [1, 24], individual effort [15], paraphrasing [9], selective test cases [1], stepwise abstraction [25], and checklists [17]. While these claims are not all mutually exclusive, they clearly reveal confusion within the community about review factors and their effectiveness.

These issues in the state of review research are not raised with the intent of denigrating the research or the researchers. Instead, they are raised to highlight the difficulty and cost of obtaining empirically-founded understanding of a highly manual, knowledge-intensive, and collaborative activity. One contribution of this research is to demonstrate how an instrumented, computer-mediated environment can resolve some of the methodological difficulties confronting current review researchers. The next section, however, presents some of the practical problems in obtaining good review outcomes with traditional manual review methods.

1.2 Issues in FTR practice

Despite the methodological issues and credit assignment variations noted above, most research tends to agree that manual review, when properly carried out, is effective. Research also tends to agree that manual review is expensive. For example, one person-year of technical staff time is required per 20 KLOC for FTR at Bell-Northern Research, and this cost adds 15-20% new overhead onto development [24]. Boeing Computer Services found reviews to be "extremely expensive" [14]. Such admissions are usually followed by analyses demonstrating that this upstream investment is more than recouped through decreases in downstream rework costs.

Although manual FTR, when properly carried out, is typically cost-effective in the long run, this is a significant qualification, since manual FTR is very difficult to properly carry out. The primary obstacles to successful practice have been documented [7, 11, 12] and include:

- *Insufficient preparation.* A ubiquitous cause of low quality review is when one or more inadequately prepared reviewers attempt to "bluff" their way through the review process. This problem is serious enough that fairly devious remedies are presented in the literature. One such approach is to deliberately leave out one page of the review materials when distributing them to participants: those who prepare will notice the absence and contact the review leader.

- *Moderator domination.* In a group meeting, it is easy for the moderator to inadvertantly or premeditatedly abuse this role by inhibiting or intimidating the other participants. This results in reduced participation and reduced review quality.

- *Incorrect review rate.* Each minute of a review meeting is intrinsically expensive, since it requires the simultaneous attendance and involvement of at least three and frequently six to eight technical staff personnel. Thus, the rate of review is critical to its cost-effectiveness: too slow and the time (and salaries) of several technical personnel is wasted; too fast and the quality of review decreases.

- *Ego-involvement and personality conflict.* The fact that one of the review member's work artifacts is under review can lead to significant interpersonnel problems.

Review always requires substantial diplomacy and care on the part of each member.

- *Issue resolution and meeting digression.* The expense of review meetings and the complexity of software dictates that review sessions not evolve into problem-solving sessions. All instructional materials we have seen cite this issue as crucial to review success, stating that reviewers must "raise issues, but don't resolve them." They also note that it requires significant reviewer effort and continual moderator vigilence to prevent such resolution-oriented discussion.

- *Recording difficulties and clerical overhead.* Manual review requires a scribe to record the outcome of the process. Capturing the information generated during a review meeting completely and accurately is extremely difficult, as noted in the literature, and as anyone who has ever attempted the role of scribe will attest. Methods involving audio-visual aids and a "telegram style" of note-taking have been proposed to support this process.

  Substantial additional clerical overhead is induced if the data collection adequate for the purposes of research on review or review process improvement is required. Published review data has only come from very large organizations able to allocate resources to this activity.

These problems are not specific to FTR, but appear in virtually all types of meeting-based group work. The same list of group process problems appears in research to motivate the design of an electronic meeting room system [21].

The previous two sections provide evidence for a central claim in our research: the current manual nature of FTR makes it difficult to effectively carry out review, and makes it difficult to measure the process and products of review in such a manner as to understand review, compare review experiences across organizations, and improve the process based upon empirical findings. The remainder of this paper presents how we are addressing these issues in our research on CSRS.

2 CSRS

CSRS is implemented using Egret [16], an environment for exploratory collaborative group work. Egret provides a multi-user, hypertext environment for Unix and X windows. Egret has a client-server architecture, where a database back-end server process written in C++ communicates over TCP/IP to client processes implemented via a customized version of Lucid Emacs.

Just as Egret is a generic framework for collaborative group work, CSRS is a generic framework for computer-mediated FTR. The current version of CSRS provides a set of language constructs for instantiation of a computer-mediated FTR method in terms of an interconnected data model and process model. This paper focusses on FTArm, the method with which we have the most experimental experience.

2.1 The FTArm Method

FTArm is a computer-mediated FTR method designed to leverage off the strengths of an on-line environment to address the problems of manual review raised in Section 1.2. The FTArm process consists of seven well-defined phases, as illustrated by the diagram in Figure 1. The FTArm method is not specific to a review artifact type or development phase.

Setup. In this phase, the moderator and/or the producer decide upon the composition of the review team and the artifacts to be reviewed. The moderator or producer then restructures the review artifact into a multi-node, interlinked hypertext document stored within the CSRS database. Regular expression-based parsing tools available in CSRS can partially or fully automate this database entry and restructuring process.

Orientation. This phase prepares the participants for the private review phase through an overview of the review artifacts. The exact nature of this overview depends upon the complexity of the review artifact and the familiarity of the reviewers with it, and can range from a simple e-mail message to a formal, face-to-face meeting.

Private review. In this phase, reviewers analyze the review artifact nodes (termed "source" nodes) privately and create issue, action and/or comment nodes. Issue and action nodes are not publicly available to other reviewers, though comment nodes are publicly available. Comment nodes allow reviewers to request clarification about the logic/algorithm of source nodes, or about the review process, and may also contain answers to these questions by other participants.

Figure 2 contains a snapshot of one reviewer's screen during the private review phase. The function `t*node-schema!combine-field-IDs` is the review artifact under analysis, as displayed in the left hand window. A checklist of defect classifications appears in the upper right window, while a defect concerning this function is being documented in the lower right window.

In FTArm, reviewers must explicitly mark each source node as reviewed when finished. While reviewers do not have access to each other's state during private review, the moderator does. This allows the moderator to monitor the

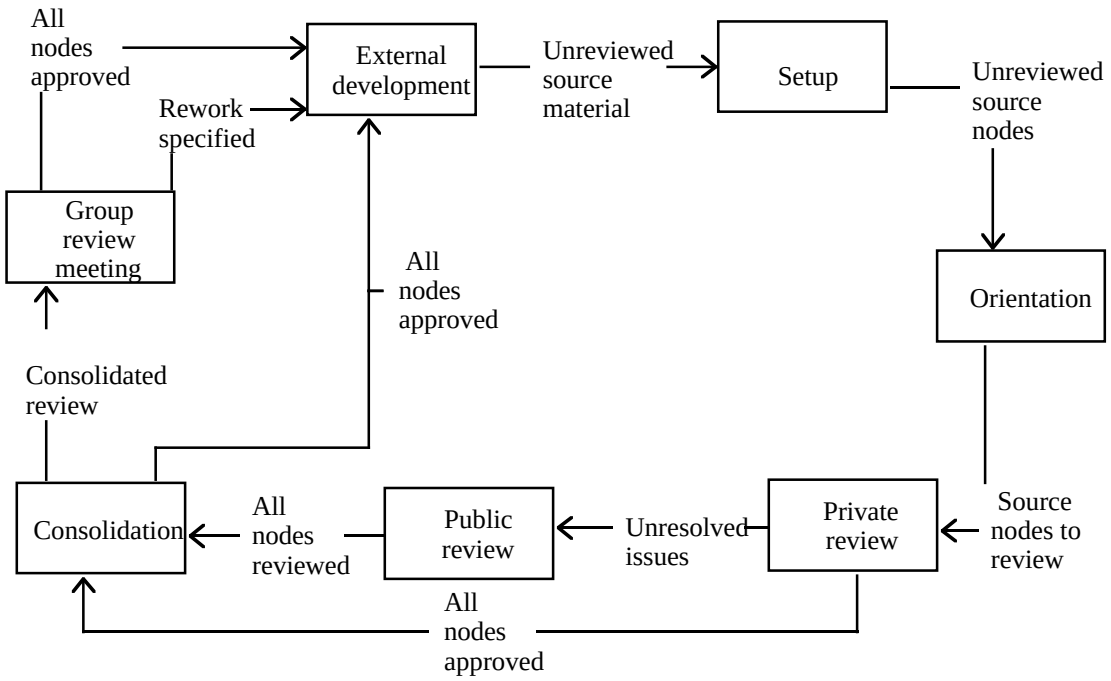

Figure 1: *The seven phases in the FTArm method, along with the primary entry condition for each phase.*

progress of private review. Private review normally terminates when all reviewers have marked all source nodes as reviewed. In the event that no reviewer has created any issues, review would terminate at this point. Otherwise, public review begins.

Public review. In this phase, all nodes are now accessable to reviewers, and all participants (including the producer) react to the issues and actions by voting (a modified Delphi process). Participants can also create new issue, action or comment nodes based upon the votes or nodes of others. Voting is used to determine the degree of agreement within the group about the validity and implications of issues and actions. Public review normally concludes when all nodes have been read by all reviewers, and when voting has stabilized on all issues.

Consolidation. In this phase, the moderator analyzes the results of public and private review, and produces a condensed written report of the review thus far. These consolidated reports are more comprehensive, detailed, and accurate than typical review reports from traditional review methods. Rather than simply a checklist of characteristics with brief comments about the general quality of the source, consolidation reports contain a re-organized and condensed presentation of the analyses provided by reviewers in issue, action, and comment nodes, thus providing contrasting opinions, the degree of consensus, and proposals for changes.

CSRS provides the moderator with various tools to support the generation of a nicely formatted LaTeX document containing the consolidated report. If the group reached consensus about all of the issues and actions during public review, then this report presents the review outcome with respect to artifact assessment. A second review outcome is the measurements of review process and products, as discussed in Section 2.2 below.

Group review meeting. If the consolidated report identifies issues or actions that were not successfully resolved via public and private review, the FTArm method requires a face-to-face, group meeting as the final phase. Here the moderator presents only the unresolved issues or actions and summarizes the differences of opinion. After discussion, the group may vote to decide them, or the moderator may unilaterally make the decision. The moderator then updates the CSRS database, noting the decisions reached during the group meeting and then generating a final consolidated report representing the product of review.

2.2 Instrumenting FTR in CSRS

Since the initial design of CSRS in 1991, research on effective instrumentation for software review has been a primary research focus. Our goal for this instrumentation is to provide a basis for empirically-based process experi-

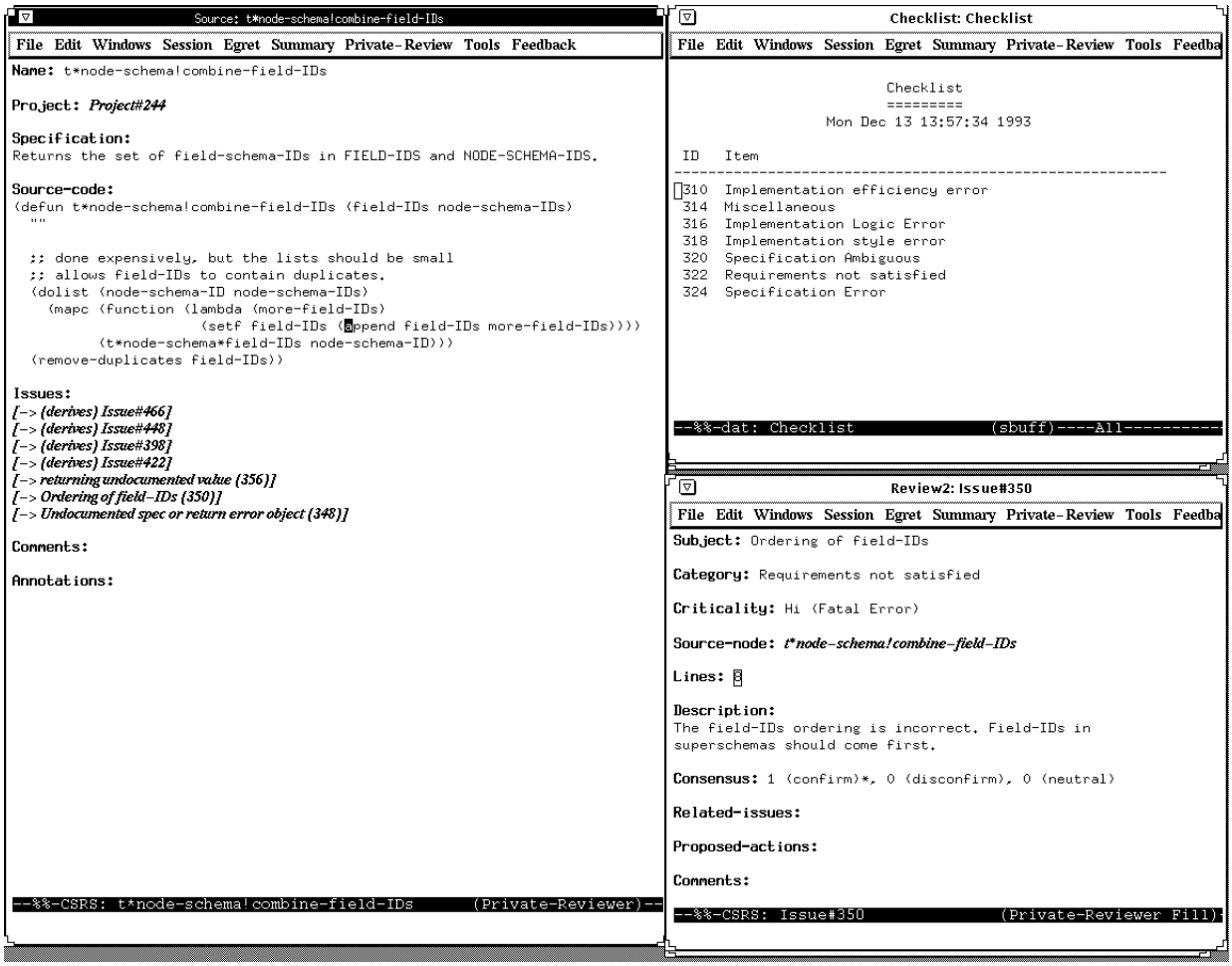

Figure 2: *A CSRS screen illustrating the generation of an issue.*

mentation and improvement.

Such instrumentation is a major feature distinguishing CSRS from other automated review environments such as ICICLE [6], Scrutiny [13], or INSPEQ [17]. CSRS supports both *outcome* and *process* instrumentation.

Outcome instrumentation. These mechanisms allow review analysts to query the CSRS database during or after review for such information as the number of nodes generated of a given type, the set of nodes containing a particular value in a particular field, or the set of nodes partaking in a specific relationship to other nodes.

Outcome measures are very useful. First, the number and severity of identified defects provides a first-order estimate of the quality of the software under review.

Second, outcome measures may suffice to reveal certain problems with the review team or review process. For example, marginally productive members of the review team might be identified (after a sufficient number of review instances) as those who contribute little, or who contribute non-productively (by simply affirming comments made by others), or who use review for political purposes. Such a method is less expensive and more objective than techniques where behavioral data (such as "Shows solidarity", "Shows Tension", and so forth) is collected by a passive observer of the review in order to provide feedback on the quality of reviewer participation [11].

Third, outcome measures can contribute to empirically-guided process improvement. For example, an organization may be able to bound certain review factors (such as the number of participants or the size of artifact to review) by measuring these factors across a large number of otherwise similar review instances, and then correlating them to review effectiveness.

Process instrumentation. These mechanisms provide insight into the *sequence* and *duration* of review activities. Analysis and application of this process instrumentation is

an exciting and intensive focus of our current research.

Process instrumentation is implemented by a general purpose timestamp subsystem in Egret, the collaborative infrastructure for CSRS. This facility enables calls at strategic points within CSRS to record the occurrence of application-specific events of interest. For example, CSRS timestamps when the user reads a node, writes a node, closes a node, traverses a link, marks a node as reviewed, and so forth. Egret provides mechanisms to cache timestamp data at the local client during a review session, and write the cache contents out to the server database at disconnect time.

One simple use of timestamp data is to calculate elapsed-time information. For example, subtracting the timestamp recorded when a user reads a particular source node from the timestamp recorded when the user closes that node yields the elapsed time during which the source node was displayed to the user.

However, this elapsed display time is not a useful measure of the *effort* spent reviewing a node, as we found out after experimental evaluation of this mechanism. In a multiple window, multi-tasking workstation environment, review using CSRS is typically interrupted by phone calls, impromptu meetings with colleagues, reading and answering e-mail, and so forth. Thus, the elapsed display time has two components: the time during which the user was actually reviewing, and the during which the node was displayed but the user's focus of attention was elsewhere.

Our process instrumentation currently detects this latter "idle time" via a bottom-up, timer-initiated timestamp process that runs in parallel with the top-down, user-triggered timestamps. Once per minute, the timer process wakes up, determines whether or not there has been any recent low-level mouse or keystroke activity within the CSRS application, and if so generates a "busy" timestamp. In our experiences thus far, this combination of top-down and bottom-up timestamping provides high quality measures of effort with a precision of plus or minus one minute.

Process instrumentation can provide substantial new insight into review. First, these process measures can combine with outcome measures to indicate the effort expended on review by each participant on *individual* source nodes, as well as the cumulative effort expended by all participants across the entire review artifact. It is well known that precise and accurate measurement of cumulative review effort (including preparation) is notoriously difficult to obtain in typical industrial settings [11]. Moreover, no other manual or automated review method provides effort data at the grain size of individual components of a review artifact. Such fine-grained, high quality data makes possible interesting new forms of experimentation, as discussed below.

3 Current results

Our results fall into two categories: design results, involving the impact of previous research on software review and collaborative work on the design of CSRS and FTArm, and empirical results, involving the measurements produced thus far by our use of the system and method.

3.1 Design results

One insight from our research with CSRS and FTArm is that introducing substantial computer support into the fundamentally manual process of traditional review creates a fundamentally different process. As a result, we take issue with research claiming to "automate" Fagan code inspection, such as the research on ICICLE [6]. While this system appears to provide a useful form of computer-supported review, the introduction of substantial computer support results in a method fundamentally different from Fagan's code inspection.

The design of CSRS and FTArm, therefore, has not been motivated by the goal of transliterating a manual "best practice" into a computer-supported form. Instead, our design attempts to implement an environment and method that exploits the known strengths of computer-supported cooperative work environments to address known problems in FTR. To illustrate, the next paragraphs discuss how the FTArm method addresses the pitfalls in manual review presented earlier in Section 1.2.

- *Insufficient preparation.* FTArm eliminates the problem of detecting insufficient preparation, since the preparation phase in traditional review corresponds to private review in FTArm, and since the activities of each reviewer during private review is precisely instrumented and known to the moderator. Of course, the method cannot force an intransigent participant to bring up the system and look at the materials, but it does eliminate the most important problem of reviewers "exaggerating" their degree of preparedness to the review leader.

- *Moderator domination.* FTArm is designed to prevent the moderator from most forms of domination that are possible in manual review. During private review, of course, all moderator influence is eliminated. During public review, the moderator can potentially influence outcome by "flaming", but cannot physically prevent contributions of other members by monopolizing air time as is possible in synchronous review. Even the group meeting phase of FTArm is less vulnerable to moderator domination, since the set of issues has already been established and documented.

- *Incorrect review rate.* Since most review in FTArm occurs asynchronously, the cost of review is much less sen-

sitive to its rate. The desire of one participant to review slowly makes no impact on the rate (or cost) of review for any other participant. Asynchronous interaction also eliminates "air-time fragmentation"—the costly idle time spent by participants in face-to-face reviews while waiting for a turn to speak.

- *Ego-involvement and personality conflict.* On-line and asynchronous review allows reviewers time to consider their choice of words carefully. During private review, in fact, it is possible to use CSRS the "day after" to modify comments that appear inappropriate, after a night's reflection, before any other reviewers see them. In general, diplomacy is much easier in a non-real time environment.

- *Issue resolution and meeting digression.* As noted previously, the constraints of a synchronous meeting context require participants to raise issues, but not to resolve them. In FTArm, the problems of cost and digression arising from issue resolution are vastly minimized or eliminated.

  More importantly, analysis of node content data from our review experiences reveals that issues are frequently difficult to articulate *except in the context of proposing an action to resolve it*, and that a natural tendency of most reviewers, immediately upon the identification of an issue, is to suggest a solution. As a result, FTArm explicitly *encourages* the interleaving of action proposal and discussion with issue proposal and dicussion during review. Our experiences suggest that capturing action proposals when the reviewer first thinks of them, rather than artificially delaying their discussion until some future meeting time, is certainly a more natural and possibly a more efficient method of review.

- *Recording difficulties and clerical overhead.* FTArm eliminates the role of scribe altogether, except for the group meeting phase where the role is considerably simplified. The CSRS system trivially resolves the problems of capturing review commentary completely and accurately. CSRS also provides the moderator with tools to support restructuring and reformatting of the data from its on-line, hypertext format into a form suitable for a linear, hardcopy presentation. Such features substantially reduce the clerical overhead normally associated with review.

  Most important from a research perspective, CSRS collects data and process measures automatically and unobtrusively. This means that research on review can be performed in any size organization with any amount of resources allocated to review.

Taken as a whole, the FTArm method implements an incrementally increasing level of both collaborative involvement and review cost. During private review, collaboration is actively prevented, and the cost of review to each participant is restricted to the expense of their personal review efforts. During public review, partial collaboration is supported by allowing each reviewer on-line access to the comments made by others, which incurs some additional cost. Finally, the review group meeting provides the highest cost collaboration, but is restricted in scope to only those issues that cannot be resolved by the previous, lower-cost forms of collaboration.

3.2 Empirical results

The second category of results involves the empirical data derived from the laboratory use of CSRS.

Review Experiments	7
Artifact Size	450–750 lines
Review Rate	200–500 lines/hr
Group Size	4–6 people
Duration	10–35 days
CSRS Sessions	2–28 logins/reviewer
Issues Generated	50–104 nodes
Issue Generation Rate	3–12 issues/hr

Figure 3: *Ranges of key review statistics.*

Figure 3 summarizes some key data concerning our review experiences during the development of CSRS and FTArm. The primary goal of these reviews were to validate the essential design characteristics of the system and to support our own quality assurance activities. However, some general observations can be made.

First, the asynchronous nature of FTArm review appears to extend the upper bound on the size of the review artifact. For larger artifacts, reviewers tend to partition the review into a larger number of sessions without loss of quality. Most manual methods require the entire artifact to be reviewed within a single meeting, which restricts the maximum size of the artifact.

Second, the on-line nature of CSRS also appears to extend the upper bound on the number of review participants. Unless all participants generate a very large number of nodes during private review, FTArm should scale unchanged to review group sizes of 10-12. Much greater increases in review group size may be possible by modifying the method instantiated in CSRS to prevent a "combinatorial node explosion." Most manual methods, in contrast, strictly limit the group size to between three and six participants.

Third, the asynchronous style of public review in FTArm can potentially lead to longer review durations than

those normally occurring in manual methods. We did not limit the duration of public review phase in our reviews, though this is a very reasonable approach to meeting a scheduled review duration.

Finally, these reviews exhibit wide variation in such measures as the total number of issues generated, the issue generation rate, and review rate. Discovering the underlying causal factors for these values and their variations motivate several on-going studies, discussed next.

3.2.1 Predictive measures of review effort

The goal of one research project is to discover predictive measures for review effort. This has been identified as a key open problem in formal technical review [10]. We are currently investigating whether or not various measures of complexity can serve as a predictive measure of review effort for source code artifacts. Three measures have been chosen for analysis: Halstead's volumetric complexity, McCabe's cyclometric complexity, and simple lines of code. Intuitively, it seems reasonable that the more complicated a program entity, the more difficult it is to understand and review, and the greater the number of issues and errors it will contain.

A sufficiently precise correlation would allow accurate estimation of time and resources required for review of source code based upon its computed complexity or conversely, accurate sizing of the artifacts to the time and resources available for review. CSRS is uniquely able to support this research, since no other review method is able to measure effort at the grain size required to assess complexity-related relationships.

So far, however, no statistically significant correlation has been observed for any of the three measures of complexity and review effort, either within or across reviewers. Values of r^2 tend to vary between 0.15 and 0.30. Interestingly, the lines of code measure of complexity is highly correlated with Halstead's volumetric complexity ($r^2 \approx 0.95$), while McCabe's cyclometric complexity was uncorrelated with any other measure of complexity or review effort.

Our initial experiences lead us to believe that either lines of code or Halstead's complexity measure can serve as a partial, complexity-based predictor of review effort, but that a high quality predictor must take more than just a single factor into account. For example, review of code produced by inexperienced developers leads to a wide spectrum of issue types, ranging from comments about the appropriate use of the programming language, to the application platform, to design aesthetics, and so forth. Review of code by experienced developers has a much narrower range of comments. We have observed that very simple functions written by inexperienced developers may engender a great deal of review activity involving these types of issues. As we accumulate more data using CSRS, we will continue to explore correlations between review effort and these and other factors.

3.2.2 Behavioral strategies during review

A second empirical investigation involves analysis of participant behavior during review as captured by the timestamp mechanism. Timestamps record the sequence and duration of reviewer actions, and provide a means to reconstruct the fine-grained strategies employed by reviewers. As a simple example, one private review strategy is to visit each source node, generate all issues relevent to that node, then move on to the next. Another strategy is to visit all source nodes first to provide an overview of the entire artifact, then selectively choose nodes for issue generation.

Identification of behavioral strategies used in CSRS will help us to improve the system by identifying bottlenecks and opportunities for additional support. It can also lead to improved understanding of what makes a system easily reviewed. Finally, it provides a means to determine when and if common review strategies are used, which is extremely useful in comparative analysis of review factors and methods.

To facilitate behavioral strategy analysis, we developed a simple visualization system called Timeplot. This system processes the thousands of timestamps generated during a review and generates a graphical representation of the contents of the reviewer screen during each minute of review, along with any idle periods for each CSRS session. Timeplot graphs allow identification of potential strategic behaviors by manually "walking through" a reviewer's behavior. These graphs also facilitate a number of advanced analytical techniques, such as phase analysis, that can be used detect and classify low-level sequences of activity as higher-level patterns.

3.2.3 Review commentary classification

Another empirical investigation concerns analysis of the types of information captured by CSRS. In one representative review, we found that approximately 80% of issues detail traditional FTR concerns such as defects. The remaining 20% provided either: (a) significant new design rationale information; (b) new clarification of the specifications or behavior of the application under development; or (c) new clarification of the specifications or behavior of the underlying infrastructure (such as the source language or operating system). These forms of knowledge are not typically captured in manual FTR—indeed, they might well be viewed as a "digression".

4 Conclusions and future directions

CSRS and FTArm demonstrate that an appropriately instrumented collaborative support environment for formal technical review can be designed to ameliorate or overcome significant obstacles to the success and efficiency of current manual FTR practice. Perhaps more importantly, however, this paper has described how such an environment can provide instrumentation that provides a wealth of high quality, useful data on the process and products of FTR. With manual methods, capturing this data is always time-consuming, expensive, and error-prone, if it can be captured at all. In CSRS, this data is captured for *free*. We hope that this paper will inspire more research and development on automated formal technical review environments, and that useful instrumentation support will form an essential part of such efforts.

One upcoming empirical study is a controlled laboratory experiment to assess the relative contributions of three examination techniques to review effectiveness and efficiency. This research, conducted as part of an upcoming doctoral dissertation, will help to clarify some of the confusion in current FTR research about the causal factors underlying review outcomes.

A second upcoming project is external validation through a select number of technology transfer experiments with industry sites. This will provide useful experience and data on the process of CSRS adoption within a variety of industrial software development organizations. We hope that future research on CSRS will be motivated through its use to improve the quality of the industry software.

Acknowledgments

The author gratefully acknowledges the other members of Collaborative Software Development Laboratory: Danu Tjahjono, Rosemary Andrada, Carleton Moore, Dadong Wan, and Robert Brewer for their contributions to the development of CSRS. Support for this research was partially provided by the National Science Foundation Research Initiation Award CCR-9110861.

References

[1] A. Frank Ackerman, Lynne S. Buchwald, and Frank H. Lewski. Software inspections: An effective verification process. *IEEE Software*, pages 31–36, May 1989.

[2] Lowell Jay Arthur. *Improving Software Quality*. Wiley Professional Computing, 1993.

[3] V.R. Basili and R.W. Selby. Comparing the effectiveness of software testing strategies. Technical Report TR-1501, University of Maryland at College Park, Department of Computer Science, 1985.

[4] V.R. Basili, R.W. Selby, and D.H. Hutchins. Experimentation in software engineering. *IEEE Transactions on Software Engineering*, SE-12(7):733–743, July 1986.

[5] David B. Bisant and James R. Lyle. A two-person inspection method to improve programming productivity. *IEEE Transactions on Software Engineering*, 15(10):1294–1304, October 1989.

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

[7] Lionel E. Deimel. *Scenes of Software Inspections. Video Dramatizations for the Classroom*. Software Engineering Institute, Carnegie Mellon University, May 1991.

[8] Robert Dunn. *Software Quality: Concepts and Plans*. Prentice Hall, 1990.

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

[10] Michael E. Fagan. Advances in software inspections. *IEEE Transactions on Software Engineering*, SE-12(7):744–751, July 1986.

[11] D. P. Freedman and G. M. Weinberg. *Handbook of Walkthroughs, Inspections and Technical Reviews*. Little, Brown, 1990.

[12] Tom Gilb and Dorothy Graham. *Software Inspection*. Addison-Wesley, 1993.

[13] John Gintell, John Arnold, Michael Houde, Jacek Kruszelnicki, Roland McKenney, and Gerard Memmi. Scrutiny: A collaborative inspection and review system. In *Proceedings of the Fourth European Software Engineering Conference*, Garwisch-Partenkirchen, Germany, September 1993.

[14] Robert L. Glass. *Modern Programming Practices: A Report from Industry*. Prentice-Hall, Englewood Cliffs, N.J., 1982.

[15] Watts S. Humphrey. *Managing the Software Process*. Addison Wesley Publishing Company Inc., 1990.

[16] Philip M. Johnson. Supporting exploratory CSCW with the EGRET framework. In *Proceedings of the 1992 Conference on Computer Supported Cooperative Work*, November 1992.

[17] John C. Knight and E. Ann Myers. Phased inspections and their implementation. *Software Engineering Notes*, 16(3):29–35, July 1991.

[18] Nancy G. Leveson and Clark S. Turner. An investigation of the Therac-25 accidents. *IEEE Computer*, 1993.

[19] Johnny Martin and W. T. Tsai. N-fold inspection: A requirement analysis technique. *Communications of the ACM*, 33(2):225–232, February 1990.

[20] G. Myers. A controlled experiment in program testing and code walkthrough/ inspection. *Communications of the ACM*, 21(9):760–768, September 1978.

[21] J. F. Nunamaker, Alan R. Dennis, Joseph S. Valacich, Douglas R. Vogel, and Joey F. George. Electronic meeting systems to support group work. *Communication of the ACM*, 34(7):42–61, July 1991.

[22] D.W. Parnas and D.M. Weiss. Active design reviews: Principles and practices. *Proceedings of Eighth International Conference on Software Engineering, London, England*, pages 132–136, August 1985.

[23] Ronald Peele. Code inspections at first union corporation. In *Proceedings of COMPSAC'82: The IEEE Computer Society's Eighth International Computer Software and Applications Conference*, pages 445–446, Silver Springs, MD., November 1982. IEEE Computer Society Press.

[24] Glen W. Russell. Experience with inspection in ultralarge-scale developments. *IEEE Software*, January 1991.

[25] R.W. Selby. *Evaluations of software technologies: Testing, CLEANROOM, and metrics*. PhD thesis, University of Maryland at College Park, Department of Computer Science, 1985.