

SUPPORTING TECHNOLOGY TRANSFER OF FORMAL TECHNICAL REVIEW THROUGH A COMPUTER SUPPORTED COLLABORATIVE REVIEW SYSTEM

Philip M. Johnson
Department of Information and Computer Sciences
University of Hawaii
Honolulu, HI 96822
(808) 956-3489
johnson@hawaii.edu

Abstract

Formal technical review (FTR) is an essential component of all modern software quality assessment, assurance, and improvement techniques, and is acknowledged to be the most cost-effective form of quality improvement when practiced effectively. However, traditional FTR methods such as inspection are very difficult to adopt in organizations: they introduce substantial new up-front costs, training, overhead, and group process obstacles. Sustained commitment from high-level management along with substantial resources is often necessary for successful technology transfer of FTR.

Since 1991, we have been designing and evaluating a series of versions of a system called CSRS: an instrumented, computer-supported cooperative work environment for formal technical review. The current version of CSRS includes an FTR method definition language, which allows organizations to design their own FTR method, and to evolve it over time. This paper describes how our approach to computer supported FTR can address some of the issues in technology transfer of FTR.

1 Introduction

Among all the software quality improvement methods currently known, formal technical review (FTR¹) enjoys unique advantages. 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 [16, 3]. 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 [1]. FTR forms an essential part of methods and models for very high quality software, such as Cleanroom Software Engineering [15] and the SEI Capability Maturity Model [18]. Finally, FTR displays a unique ability to improve the quality of the producer as well as the quality of the product by dispersing knowledge about applications and

¹We define *formal technical review* as “a structured encounter where a group of technical personnel analyzes an artifact to improve quality. The analysis produces a structured artifact that assesses or improves the quality of the artifact as well as the quality of the method.” This definition includes methods such as Fagan’s code inspection [6, 7], Phased Inspections [14], and FTArm (discussed here), but excludes methods such as informal peer reviews and walkthroughs.

development skills across the organization.

Given the range of advantages ascribed to FTR, and the substantial improvements in quality and cost-reductions attributed to it by some organizations, it is curious that formal technical review is not ubiquitous in modern software development. Although rigorous data on industrial use of FTR is not publically available, responses by 70 participants to an informal survey we conducted on FTR via USENET revealed that FTR is practiced irregularly or not at all in over 80% of the surveyed organizations. Similar non-rigorous evidence for a low level of FTR adoption in industry is discussed in [4].

Since 1991, we have been designing and experimentally evaluating a computer-supported cooperative work environment for FTR called CSRS [11, 12, 13]. One product of this research was the creation of a new, highly instrumented, asynchronous review method called FTArm that addresses a multiplicity of problems arising in the research on and practice of traditional FTR. As we began discussing technology transfer of CSRS and FTArm with industrial organizations, we became aware of a spectrum of organizational issues surrounding the technology transfer of FTR in general and CSRS in particular that must also be addressed.

These issues and others motivated a recent redesign of CSRS to provide a specialized process modelling language for FTR. The language is intended to allow organizations to design their own FTR method for use with CSRS, and to support incremental evolution in the method as the organization’s needs for and use of FTR changes.

In the next section, we present some of the problems involved in successful technology transfer and adoption of FTR. The following section briefly overviews the CSRS system and the FTArm method. Following this we discuss how the process modelling facilities of FTArm can be applied to address some of the problems that arise in technology transfer and adoption of FTR.

2 Issues in FTR Technology Transfer

2.1 The transfer process

Our model of technology transfer follows research which does not view it as a “transfer” at all, but rather as a *reconstruction* by one organization of knowledge, expertise, and technology generated by another [5]. This contrasts with the conventional view of technology trans-

fer, in which the technology is viewed as a relatively static object whose successful transfer induces a change in the receiving organization without impacting upon the technology itself. When participants in the transfer process interpret the technology differently, the conventional view holds that they are either misperceiving the technology or the technology has been somehow distorted.

The conventional view appears occasionally in the literature on industrial use of FTR methods, such as Fagan's code inspection. Adoption failures are here interpreted as either a misperception of the meaning of the method or a failure to implement all parts of the method. Successful technology transfer, from the perspective of this literature, is simply a matter of total adherence and commitment to a single approach to formal technical review.

Other FTR literature describes a more context-sensitive and reconstructionist view of the adoption process. For example, a study of FTR technology transfer at Hewlett-Packard reveals that FTR adoption goes through a series of stages and that blind adherence to a single standardized process is a recipe for failure, not success [8]. The four stages observed at Hewlett-Packard are described in this study in the following way:

- *Experimental.* This stage is characterized by trial adoption of a not well understood technique by a few groups within the organization with relatively little institutional support. Surviving the experimental stage of technology adoption appears to depend upon: (a) visionary people who can look at tools and process from another context and see how they can be applied locally; (b) management support for visionary attempts without penalty for failure; and (c) a supportive infrastructure, since mistakes and failures will occur and early success is very fragile from an organizational standpoint.
- *Initial Guidelines.* Progression out of the experimental phase is marked by the appearance of training classes and educational materials on the technique, and the creation of small-scale infrastructure within the organization to promote the technology. However, the Hewlett-Packard researchers caution that readily available training is a necessary but not sufficient condition for technology dispersion. For the technology to become further incorporated into the organization, effort must be made to communicate success with the method throughout the organization, through activities such as newsletters, conferences, and so forth. In addition, high-level management must be educated in the evolving "best practice" of the method and they must continue to display commitment and allocate resources to the technology
- *Widespread Belief and Adoption.* This stage is characterized by widespread acceptance within the organization that the technology is useful and important to the organization's success. However, such "widespread belief" does not translate automatically into optimal or even effective use of the technology, and may even sow the seeds of the technology transfer's destruction.

One potential problem at this stage is that management

may become convinced that there is "one best way" and begin pushing for its total and exclusive adoption. Hewlett-Packard found that their internal divisions resisted this, preferring a consulting approach whereby corporate resources were applied to understanding the specific context and problems of a division, and then developing an individualized strategy to help the group improve their current practice.

A second potential problem is that as the use of the technology spreads across the organization, the aggregate cost of the technology to the organization becomes increasingly substantial and significant. An effective business case must be created to ensure that the technology continues to be used beyond a trial period. Otherwise management may decide that the technology, though promising, is not cost-effective when scaled to the organizational level.

- *Standardization.* While Hewlett-Packard has not yet progressed beyond the previous stage, their researchers suggest that there is a phase beyond it. This phase appears to be characterized by total integration of the technology into the organization, such that questions of appropriateness are no longer asked—the technology has become part of what makes the organization what it is. The HP researchers explicitly note that terming this stage "standardization" does not imply adherence by the entire company to a single process, but rather that every project would use some form of FTR technology in an efficient, cost-effective manner.

2.2 Obstacles to FTR adoption

Many studies assert that an FTR such as Fagan's inspection is cost-effective and improves software quality, once successfully adopted and when practiced effectively. However, it is also clear from studies that FTR is difficult to adopt and practice effectively. The following obstacles to effective FTR adoption and use is drawn from [2, 4, 19]:

- *Low Technology.* FTR typically involves a lot of "meticulous, pain-staking, manual work". Software developers, used to e-mail and on-line discussions, may resist returning to hand-written notes and extensive meetings with high clerical and administrative overhead.
- *Ambiguity in Data/Process Model.* Without proper training, manual FTR methods are easy to misinterpret or misapply in practice. Successful introduction of a method requires training to impart a precise understanding of the process to be followed, since different approaches may have widely varying benefits.

Ambiguity in the FTR method may also lead management to block introduction of a new method based upon the mistaken notion that "we've tried this before and it didn't work."

- *Absolute Expense.* The cost of developing manual FTR infrastructure (planning, training, developing forms)

and the cost of performing FTR (preparation, meetings, filling out forms, data analysis) is substantial. At Bell-Northern Research, one person-year of effort was expended for each 20 KLOC under FTR, and this introduced 15-25% new overhead to the development process. (These upstream costs were recovered downstream during testing and maintenance.)

- *Relative Expense.* An organization which already has an informal review method in place may not feel that the additional benefits will justify the additional cost.
- *Demand for proof.* Introduction of FTR requires approval from management who will want to see evidence that the process is worth the investment. However, management may frequently reject evidence from published reports as not relevant to their organization, and collecting statistically meaningful in-house data is impossible until inspections have actually been adopted.
- *Developer inertia.* Adopting FTR requires convincing management that the organization will benefit. However, adopting FTR also requires convincing developers that their professional quality of life will improve. FTR is often resisted by developers, who see it as yet one more hurdle placed between them and successful discharge of their responsibilities.
- *Training.* Successful adoption of FTR typically requires extensive training. For example, Bell-Northern Research developed a self-study video course involving examples of both effective and ineffective inspection meetings, the meeting process, team roles, error reporting, planning guidelines, and paraphrasing, along with an example program for practice. Hewlett-Packard provided both initial training and a “continuing education” program to keep participants abreast of changes and improvements to their FTR program.
- *Fire fighting.* If a project is already having process problems, then the group may be resistant to the introduction of any new process hurdles. Moreover, if the project is having extensive process problems, then FTR may not be the most important way for the team to expend their time and resources. The Capability Maturity Model, for example, only mentions peer reviews starting at Level 3, after more basic project management mechanisms have been put into practice.
- *Improved quality not beneficial to bottom line.* Quality in a certain product may be desired but is traded off against other goals (such as profit, schedule, etc.) Adoption of FTR may be resisted because it is viewed as improving quality at the expense of other goals more important to management.
- *Perceived long-term inefficiency during maintenance.* The cost of FTR during maintenance becomes particularly high, since a change to a unit requires a complete re-inspection with almost no savings from previous inspections. In contrast, re-testing a changed unit is virtually free since it simply requires re-running tests. In

maintenance-heavy contexts, management may view development of regression test suites as more cost-effective than FTR.

- *Possible ineffectiveness during maintenance.* During maintenance, change to one part of the system may have a ripple effect that causes a fault in a different part of the system. This kind of fault is extremely difficult to detect using FTR, unless the organization is willing to allow a single change to the system to trigger a massive re-inspection of all potentially affected parts. Again, testing may be viewed as more cost-effective, since testing is potentially capable of detecting such interactions.

Having now identified some of the major issues in technology transfer of FTR, we now overview CSRS, our computer supported environment for FTR. Following this introduction, we will describe how the process modelling language for FTR in CSRS can be used to facilitate the technology transfer process.

3 CSRS: Computer Supported FTR Definition and Enactment

CSRS is a multi-user, interactive hypertext environment for performing FTR, implemented using the Egret collaborative work environment [10]. Egret has a client-server architecture, where a database back-end server Unix process written in C++ communicates over TCP/IP to X window client processes implemented using a customized version of Lucid Emacs.

Egret is designed to support collaborative systems containing a mixture of interactive “user” processes directly controlled by people and autonomous “agent” processes that provide computational services. For example, many CSRS methods include a mailer agent that wakes up once or twice a day, inspects the state of review, and sends an e-mail message to participants notifying them of any new activities for them to perform.

Just as Egret is a generic framework for collaborative group work, CSRS is a generic framework for computer-supported FTR. The implementation of this generic framework involves a set of language constructs for defining a computer-mediated FTR method. To illustrate the capabilities of CSRS, the next section overviews FTArm, one of the FTR methods that can be defined using CSRS.

3.1 The FTArm Method

FTArm is a computer-mediated FTR method designed to leverage off the strengths of an on-line environment to address certain problems of manual FTR. The FTArm process consists of seven phases where participants interact within the roles of moderator, producer, reviewer, or administrator. 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 1 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 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 accessible 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 detailed and accurate measurements of review outcome and process.

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.

Process Improvement Meta-Phase. The preceding phases provide a framework for the FTR process, but also allow for evolution in response to various measurements automatically provided by CSRS. The system automatically generates a timestamped log of the sequence of nodes visited and links traversed by participants during review. CSRS analysis tools use this data to provide useful process measurements, such as the number of minutes spent by each reviewer on each source artifact, the number of issues raised per minute of review time, the review strategy employed by participants, the level of consensus in the review, and so forth. This data can be used to improve such method variables as: artifact size and complexity, review team size and composition, private review analysis technique, review checklist composition (if checklists are used), public review scope and duration, individual and team effort, and scope and duration of group review.

3.2 Process modelling in CSRS

As the preceding description illustrates, FTR in CSRS consists of structured interactions between a group of people with well-defined roles. These interactions are divided into a sequence of phases, during which they analyze source artifacts using various analysis tools and artifacts, resulting in the production of review artifacts. The process is measured and the measurement data is analyzed in various ways to provide insight into the process and products of review. Each of these fundamental characteristics

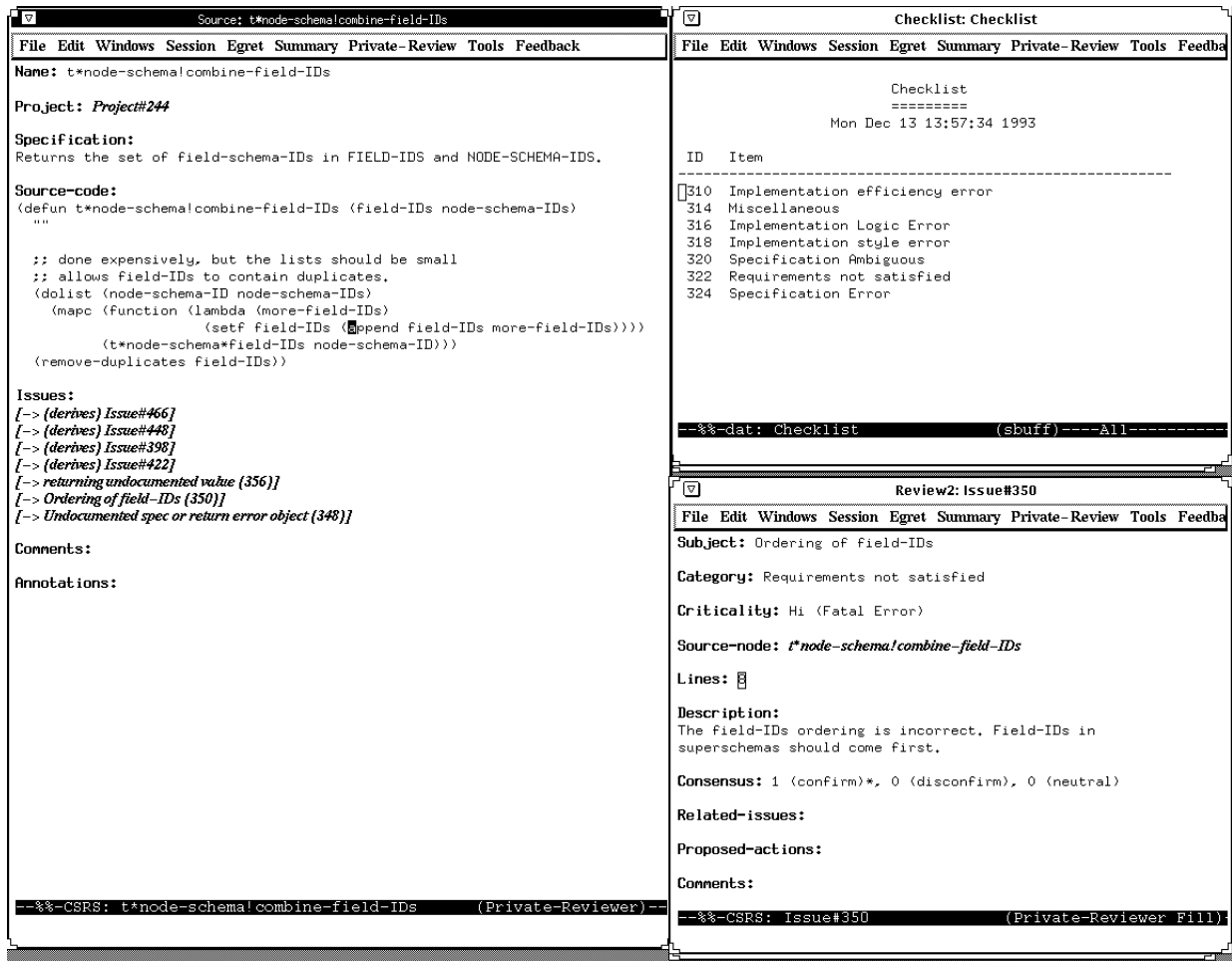


Figure 1: A CSRS screen illustrating the generation of an issue.

of an FTR method can be customized in CSRS using its process modelling language:

- **Method definition.** CSRS provides two language constructs called define-method and define-phase that allow method designers to implement a new method as a sequence of phases. CSRS does not currently support iteration or conditionals in phase sequencing, in keeping with manual FTR techniques. (However, iterative or conditional activities may occur within a phase). Phases can have entry and exit conditions associated with them to determine when it is possible to transition from one phase to the next.

Each phase also has a distinct set of operations associated with it, as specified using the define-operation construct. Each operation can be made highly context-sensitive to the current state of review and the current role of the participant.

- **User definition.** CSRS allows specification of review method roles through the define-role construct, and the actual people involved in a particular review through the define-participant construct. A given participant can play multiple roles during review. There is one hard-wired role called Administrator that is present by

default in all review methods, and which is used to bootstrap the system. Roles found in other methods can be defined in CSRS, such as moderator, producer, reviewer, and reader.

- **Artifact definition.** CSRS provides constructs such as define-node-schema, define-field-schema, and define-link-schema to allow construction of the type-level characteristics of the artifacts manipulated during review. For example, a requirements review method might provide nodes or fields of type overview, product-functions, functional-requirements, external-interface, performance-constraints, hardware-limitations, and so forth. A C++ code review method might provide nodes or fields of type class-declaration, member-function, member-variable, private-part, public-part, protected-part, template, and so forth.

The define-checklist and define-checkitem constructs allow specification of materials used during analysis and their properties, such as whether the reviewers are required to explicitly mark an item as satisfied by the artifact or not.

- **Measurement definition.** In the current version of CSRS, method designers have only binary control over

the generation of the timestamped log file containing fine-grained raw data on the activities of reviewers: either fine-grained raw data is collected or it is not. However, designers have a great deal of control over how this data is analyzed, and whether the identities of reviewers are revealed in the analysis, whether only aggregate group statistics are generated, and so forth. CSRS provides several predefined analysis tools for use by designers which generate spreadsheet-compatible data files on the review process and outcomes.

This description covers approximately half of the language constructs, and leaves out entirely those constructs concerned with user interface features of the method, such as the layout of screens and menus.

While the language provides substantial support for definition of FTR methods, it does not trivialize it. For example, the definition of FTArm in the process modelling language requires over 100 construct invocations and over 1000 lines of code. The language contains a mixture of declarative and procedural specifications of behavior, and certain behaviors (such as the agents) must be specified in terms of underlying Egret primitives. However, this definition of FTArm represents less than 3% of the total size of the Egret/CSRS system, so substantial reuse of code and development effort is achieved.

4 Technology transfer using CSRS

As mentioned previously, when we began discussing technology transfer of a previous version of CSRS (that did not include a process modelling language, but which did include FTArm), we ran into a variety of adoption problems.

First, managers were concerned that FTArm, while interesting, was either too complicated a method to implement initially or too different from their current FTR practice, and would cause significant organizational perturbations.

Second, developers were concerned about the detailed data to be gathered: the idea that their activities were being so precisely monitored caused repeated reference to the “Big Brother” [17] nature of the system, with its attendant possibility of management abuse.

Finally, there was a class of problems raised by both managers and developers concerning the use of a computer-supported cooperative work system to replace a manual process. Such “groupware technology transfer” problems are significant. The challenges of successful groupware adoption in comparison to single-user, off-the-shelf applications has been described as follows:

A word processor that is immediately liked by one in five prospective customers and disliked by the rest could be a big success. A groupware application to support teams of five nurses that initially appeals to only one nurse in five is a big disaster. [9]

Interestingly, there is a great deal of overlap between the

technology transfer problems of groupware and that of FTR. An FTR method that initially appeals to only one developer in five is also a big disaster.

From our discussions, we concluded that both managers and developers felt that computer supported FTR can significantly improve their software quality improvement practice, but that they require more control over the FTR method enacted by the environment. To provide this control, we redesigned CSRS to provide the process modelling language described above. We now illustrate how a transfer process based upon incremental evolution in the FTR method can help support adoption of FTR in general and CSRS in particular within an organization.

4.1 Experimental Phase

The initial, experimental phase of technology transfer is particularly fragile and prone to failure. During this phase, the process model enacted by CSRS must be simple, training should be minimal, and the user interface should be simple and intuitive. In addition, the use of a computer-supported tool should provide both managers and developers with significant benefits. An immediate advantage that CSRS has over traditional FTR methods is that it is a “high-tech”, on-line system, which eliminates the vast majority of clerical, manual activities formerly associated with review.

The most appropriate initial review method to enact within CSRS during the experimental phase always depends upon specific organizational factors, but perhaps the most important one is the organization’s prior experience with review.

4.1.1 No Prior FTR Experience

A common situation is one in which the organization recognizes a need to perform FTR, but has very little experience with it. In this situation, two technology transfers must occur: transfer of FTR into the organizational process, and transfer of CSRS as a technology for enactment of FTR.

Some of the obstacles to FTR adoption in organizations new to review are fears that FTR will take too much time away from coding, that it will be too expensive, and that developers will have difficulties adjusting to the increased visibility of their intermediate work products.

To explore how CSRS can support FTR adoption in this situation, we designed an FTR method called “Hello-World” (HW). The HW method consists of a single review phase in which all participants scan the review artifact and generate anonymous comments to raise and react to issues. Once all reviewers have scanned the artifact, they attend a meeting to decide upon the validity of the issues.

The HW method provides only three automated services:

1. HW provides a mailer agent that runs once a night

during review. It sends a daily “CSRS News” e-mail message to each reviewer as long as work remains to be done by the reviewer—either remaining review artifacts or new comment nodes that the reviewer has not yet seen.

2. HW provides a hard-copy mechanism to provide a nicely formatted version of the review artifact and all the comments generated about it. This artifact also provides checkboxes for each issue to log the decision of the participants at the group meeting about its validity.
3. HW provides coarse, anonymous statistics: the total number of minutes users were on-line in the system, and the total number of issues raised.

The goal of the HW method is to demonstrate to an organization new to review that FTR can be cost-effective, that review need not be excessively time-consuming or tedious, and that making work products visible in this manner can be positive for both the producer and the reviewers. By making all reviews public immediately, it stimulates activity in the system since there will be frequently a new comment to look at. Allowing reviewers to see comments in advance can reduce the chances of long or unproductive meetings, and automatic hard-copy generation and statistics keeps clerical overhead to a minimum. For management, HW provides simple statistics which can show some initial data on the ability of FTR to detect faults and the effort required to do so.

The HW method, while technically a formal technical review method, is still relatively unstructured, and does not allow much of the sophisticated process control and analysis supported in a method like FTArm. However, it appears to be much better suited than a method like FTArm for the initial buy-in phase of FTR adoption for organizations with no prior FTR experience.

4.1.2 Prior FTR Experience

The strategy for organizations with prior experience in FTR is quite different. In this case, it is important to assess the current state of FTR within the organization and tailor a CSRS method in response.

For example, an organization may have adopted a traditional inspection-based process and may have experienced some success with it, but is still in the “experimental” stage. This could be result from inconsistent application of the method resulting in inconsistent outcomes. It could also result from a lack of resources to collect data with which to publicize the successes of the method.

In this case, it might be appropriate to design a CSRS-based method that closely parallels the current method in place. Several advantages result from this strategy. First, it reduces training, since developers can be told that the CSRS FTR method is “just like” the one they are now using, except for a small set of differences. Second, the CSRS method can reduce variation in the practice of the review method, which may lead to more consistent outcomes. Finally, a CSRS-based system can automatically

collect and analyze process and outcome measurements, allowing groups with limited resources to publicize detailed and accurate data about their experiences. Such data can help obtain management commitment to FTR.

CSRS can also apply to organizations that have tried but failed to adopt traditional FTR methods. For example, one organization we work with requires review of change-request documents by at least a dozen personnel from different departments, and sometimes as many as twenty or thirty. All traditional formal technical review methods of which we are aware simply view this as an “error”: it is simply not possible to perform FTR with more than 6-8 people. Unfortunately for traditional FTR methods, the reality in this organization is that the change-request document *must* be reviewed by large numbers of people. Although an ingenious, email-based approach was developed in this organization to support FTR, we believe that many organizations in this situation would simply abandon FTR as inappropriate.

In such situations, CSRS may provide a means to expand the boundaries of FTR application. An on-line system such as CSRS can easily support reviews involving 40 or 50 reviewers who may be geographically dispersed and unable to attend face-to-face meetings. Unlike email, CSRS can provide a more structured process as well as accurate and precise measurements of process and outcome.

4.2 Later phases

We expect that technology transfer of CSRS-based FTR to an organization will progress in much the same way as the four stage model used to characterize the Hewlett-Packard adoption process. Progression beyond experimental usage involves the creation of small-scale infrastructure within the organization to further spread the technology through training. With sufficient time, the technology could become widely adopted and eventually become a standard part of the organization’s quality improvement practice.

A significant difference between CSRS-based FTR technology transfer and traditional practice is the focus upon explicit evolution in FTR method as an active part of the technology transfer process. This contrasts to a “big bang” approach, in which a single, full-blown method is taught to groups and instituted in its total, mature format. From our prior experiences discussing technology transfer with organizations, we believe that successful adoption of CSRS must be performed incrementally. The initial method must be modest in scope, but still cost-effective. As the organization becomes more used to a computer-supported method, ideas for more sophisticated services and process details will arise naturally as the organization assesses its practice. This evolutionary process of method refinement and improvement reifies the reconstructive nature of technology transfer using CSRS.

5 Conclusions

This paper presents an approach to technology transfer of formal technical review that is based upon an on-line,

interactive environment and evolution in the method as the nature of FTR within the organization matures over time.

The approach can address several of the obstacles identified above. It overcomes the “low technology” obstacle with a computer-supported approach to eliminate much of the clerical overhead involved in traditional FTR. It reduces ambiguity in method application, leading to clearer relationships between method and outcome. It reduces several cost factors for FTR. In particular, it automates collection and analysis of several common review metrics, allowing groups to quickly generate in-house data at with little additional overhead. By appropriate method definition, it can help overcome developer inertia by providing them with a tool to quickly and efficiently communicate skills and knowledge within the group.

However, the approach is not a panacea. It still requires training, resources, and a commitment to the process by management. As with other forms of FTR, it may not be appropriate for groups without basic project management mechanisms in place, and it does not resolve important maintenance-related issues.

6 Acknowledgements

The author gratefully acknowledges current and past members of the 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] Lowell Jay Arthur. *Improving Software Quality*. Wiley Professional Computing, 1993.
- [2] Victor Basili, Michael Daskalantonakis, and Robert Yacobellis. Technology transfer at Motorola. *IEEE Software*, 11(4), March 1994.
- [3] 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.
- [4] Bill Brykczynski, Reginald Meeson, and David A. Wheeler. Software inspection: Eliminating software defects. In *Proceedings of the Sixth Annual Software Technology Conference*, Alexandria, VA., May 1994.
- [5] Stephen Doheny-Farina. *Rhetoric, Innovation, Technology: Case Studies of Technical Communication in Technology Transfers*. MIT Press, 1992.
- [6] Michael E. Fagan. Design and code inspections to reduce errors in program development. *IBM System Journal*, 15(3):182–211, 1976.
- [7] Michael E. Fagan. Advances in software inspections. *IEEE Transactions on Software Engineering*, SE-12(7):744–751, July 1986.
- [8] Robert Grady and Tom Van Slack. Key lessons in achieving widespread inspection use. *IEEE Software*, 11(4), July 1994.
- [9] Jonathan Grudin. Groupware and social dynamics: Eight challenges for developers. *Communications of the ACM*, 37(1):92–105, January 1994.
- [10] Philip M. Johnson. Supporting exploratory CSCW with the EGRET framework. In *Proceedings of the 1992 Conference on Computer Supported Cooperative Work*, November 1992.
- [11] Philip M. Johnson. An instrumented approach to improving software quality through formal technical review. In *Proceedings of the 16th International Conference on Software Engineering*, May 1994.
- [12] Philip M. Johnson and Danu Tjahjono. Improving software quality through computer supported collaborative review. In *Proceedings of the Third European Conference on Computer Supported Cooperative Work*, September 1993.
- [13] Philip M. Johnson, Danu Tjahjono, Dadong Wan, and Robert Brewer. Experiences with CSRS: An instrumented software review environment. In *Proceedings of the Pacific Northwest Software Quality Conference*, 1993.
- [14] John C. Knight and E. Ann Myers. An improved inspection technique. *Communications of The ACM*, 11(11):51–61, November 1993.
- [15] Richard C. Linger. Cleanroom software engineering for zero-defect software. In *Proceedings of the 15th International Conference on Software Engineering*, 1993.
- [16] G. Myers. A controlled experiment in program testing and code walkthrough/inspection. *Communications of the ACM*, 21(9):760–768, September 1978.
- [17] George Orwell. *Nineteen eighty-four*. Clarendon Press, New York, 1984.
- [18] Mark C. Paulk, Bill Curtis, and Mary Beth Chrissis. Capability maturity model, Version 1.1. *IEEE Software*, 10(4), July 1993.
- [19] Glen W. Russell. Experience with inspection in ultralarge-scale developments. *IEEE Software*, January 1991.