Design for Instrumentation: High Quality Measurement of Formal Technical Review

Philip Johnson
Department of Information and Computer Sciences
University of Hawaii
Honolulu, HI 96822
(808) 956-3489
(808) 956-3548 (fax)
johnson@hawaii.edu

November 25, 1994

Keywords: Formal technical review, computer-supported cooperative work, software quality assurance.

Abstract

All current software quality assurance methods incorporate some form of formal technical review (FTR), because structured analysis of software artifacts by a team of skilled technical personnel has demonstrated ability to improve quality. However, FTR methods come in a wide variety of forms with varying effectiveness, incur significant overhead on technical staff, and have little computer support. Measurements of these FTR methods are coarse-grained, frequently low quality, and expensive to obtain.

This paper describes CSRS, a highly instrumented, computer-supported system for formal technical review, and shows how it is designed to collect high quality, fine-grained measures of FTR process and products automatically. The paper also discusses some results from over one year of experimentation with CSRS; describes how CSRS improves current process improvement approaches to FTR; and overviews several novel research projects on FTR that are made possible by this system.

To appear in Software Quality Journal.

Contents

1 Introduction

Formal technical review (FTR) 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 anomalies, 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 [5], to Active Design Reviews [20], to Phased Inspections [17].

Past research shows that FTR provides unique and important benefits. Some studies provide evidence that FTR can be more effective at catching errors than testing, and that it can discover different types of errors than testing [2].

In concert with other process improvements, Fujitsu finds FTR to be so effective at removing defects that they have dropped system testing from their software development procedure [1]. FTR is an integral part of all current methods intended to produce very high quality software, such as Cleanroom Software Engineering, and is present at all higher levels of the SEI Capability Maturity Model [21].

Although the software engineering research literature contains many FTR "success stories", it is by no means ubiquitous within industrial software engineering practice. In fact, evidence suggests quite the opposite. For example, in an informal survey of USENET readers on FTR, approximately 80% of 70 respondents replied that FTR is practiced irregularly or not at all within their organization.

We believe that the poor rate of adoption of FTR within industry has its roots in two fundamental properties of current FTR practice: it is manual, and it is not measured. These two properties are actually related: because FTR is manual, it is almost always too difficult and expensive to measure effectively. And because FTR is not measured effectively, it is difficult to justify to management, difficult to justify to technical staff, and difficult to customize to facilitate successful adoption and cost-effectiveness within an organization's particular development style and application domains.

As evidence, the first "lesson learned" from 6000 manual FTR experiences at Bull HN Information Systems is that precise measurements are usually too difficult and expensive to collect [23]. As Weller puts it, "you may have to sacrifice some data accuracy to make data collection easier."

In this paper, we describe our current research findings on highly instrumented, computer-supported formal technical review. The first "lesson learned" from our experiences is that you do *not* need to sacrifice data accuracy to make data collection easier, because with a computer-supported review system, highly accurate data can be collected automatically.

We have demonstrated this through the design, implementation, and evaluation of CSRS[1], a collaborative environment for formal technical review [14, 15]. CSRS contains novel instrumentation for precise, high quality measurements of FTR process and products, along with analysis tools to facilitate process improvement based upon these empirical findings. The studies reported in this article were based upon one specific FTR method defined within CSRS, called FTArm[2]. However, CSRS provides a method definition language that allows it to enact a wide range of review methods. Over the past two years, we have carried out laboratory studies of FTR using CSRS that we have analyzed to improve the environment itself and to gain new insight into FTR. Currently, we are readying the environment, instrumentation, and analysis tools for guided technology transfer into a select number of industry sites.

This paper discusses how to "design for instrumentation" in the domain of formal technical

[1] Computer-Supported Review System

[2] Formal, Technical, Asynchronous review method

review. It describes how requiring high quality measurement impacts upon the nature and scope of our system, as well as how this requirement led to qualitatively different and better insight into our computer-mediated group process.

Section 2 orients the reader to our system via selected excerpts from a recent FTR experience using CSRS. (Appendix A provides more details about the CSRS data and process model.) Section 3 describes the CSRS instrumentation with examples of the measurements of FTR process and products collected by CSRS. Section 4 describes how these measurements are utilized to generate new knowledge about FTR and devise improvements in its process and products. Section 5 discusses related research, and Section 6 summarizes the major lessons we have learned from our efforts to design for instrumentation.

2 A Reviewer's Perspective of CSRS

With the FTArm method, formal technical review using CSRS is an almost entirely on-line, asynchronous process. Review typically begins with the downloading of the review artifacts, such as requirements specifications, designs, source code, test plans, and so forth from their ASCII files into the CSRS database. CSRS semi-automatically re-structures the artifacts as a fine-grained hypertext network consisting of typed nodes and links. For example, for source code artifacts, each function and variable definition would be stored within its own node, with hypertext links to related nodes. For a requirements specification, CSRS would re-structure the document hierarchically into a set of nodes for each major section, each containing subnodes for individual specifications typed according to their focus.

After an orientation session, reviewers begin to *privately* analyze the on-line artifact. Figure 1 shows the initial screen that appears for the exemplary review experience, called Type-I, that will be used throughout this paper[3]. The left hand side of the screen contains a window with a list of review artifact nodes (in this case, source code variable, function, and macro definitions). Reviewers "middle-click" over an artifact name to retrieve it for review. Reviewers then analyze the artifact for quality-related problems and create links to comment, issue, or action nodes as necessary.

Figure 2 illustrates the screen configuration during issue generation. The artifact generating the issue appears on the left hand side of the screen, the top right hand side contains a list of issue types for this particular review process, and the bottom right hand side contains the text of the issue. A link connects this particular review artifact to the issue it generated, with anchors (in bold, italicized font) displayed in both the source node window and the issue node window. Reviewers typically traverse the network by middle-clicking on a link anchor to retrieve that corresponding node.

During the private review phase, reviewers are only permitted by the system to view the issues they themselves create. This prevents "free-loading", and enables reviewers to analyze the artifact without the bias of other reviewer's viewpoints. As reviewers work, CSRS unobtrusively generates a timestamped log file containing a listing of semantically meaningful actions taken by each reviewer. CSRS records any idle-time detected during review in the log file. These measurements will be described in more detail in Section 3.

Once reviewers have completed private review, they begin a *public* review phase. During public review, the focus of effort shifts away from the original artifact to the commentary generated by the reviewers during private review. The goal of this phase is to generate consensus about the validity,

[3] This FTR occurred during Fall of 1993 as a normal part of the software quality improvement practices of our development group. The examples are presented "as is", with no attempt to clean up misspellings or hacker's jargon.

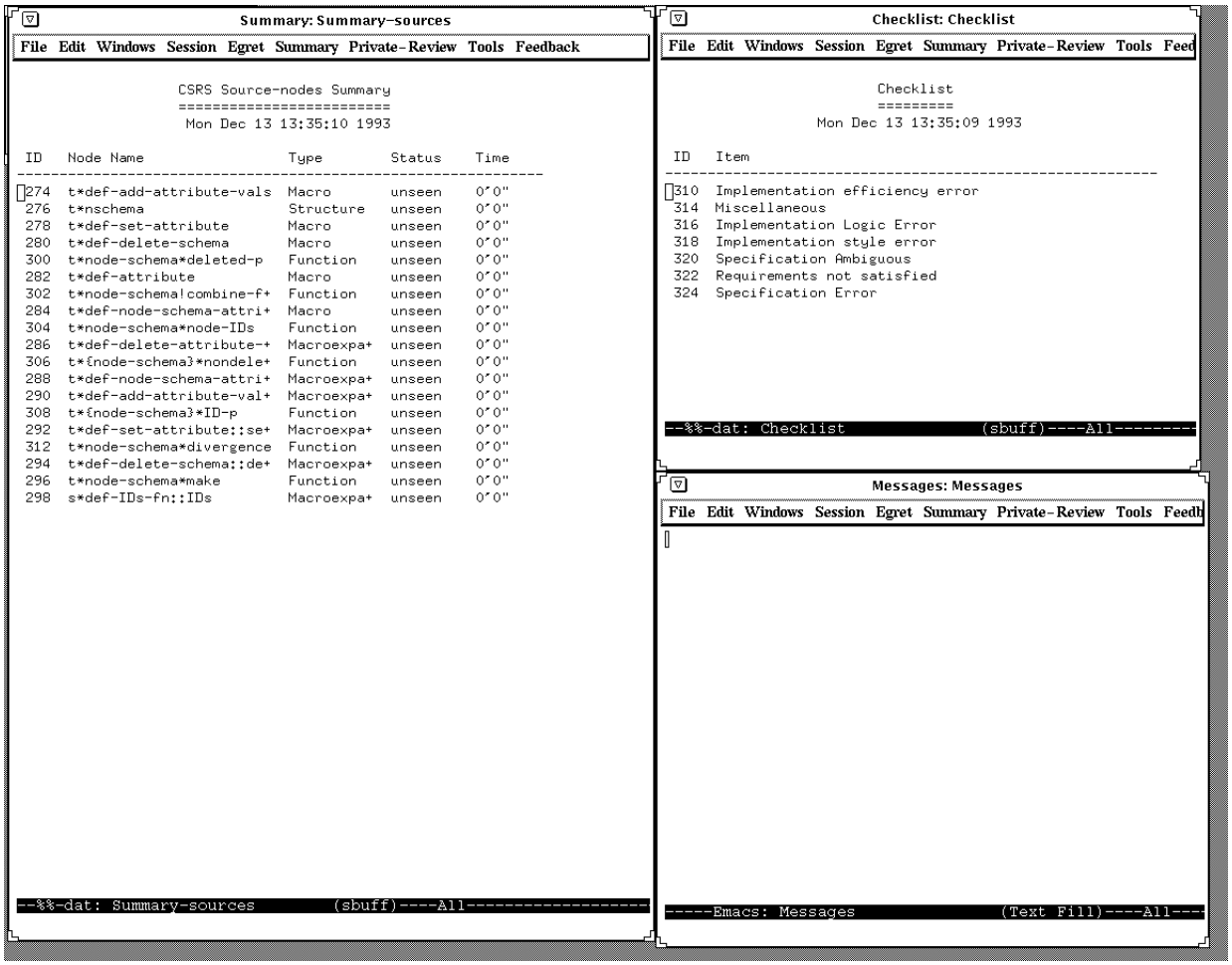

Figure 1: The initial CSRS screen when a reviewer starts private review.

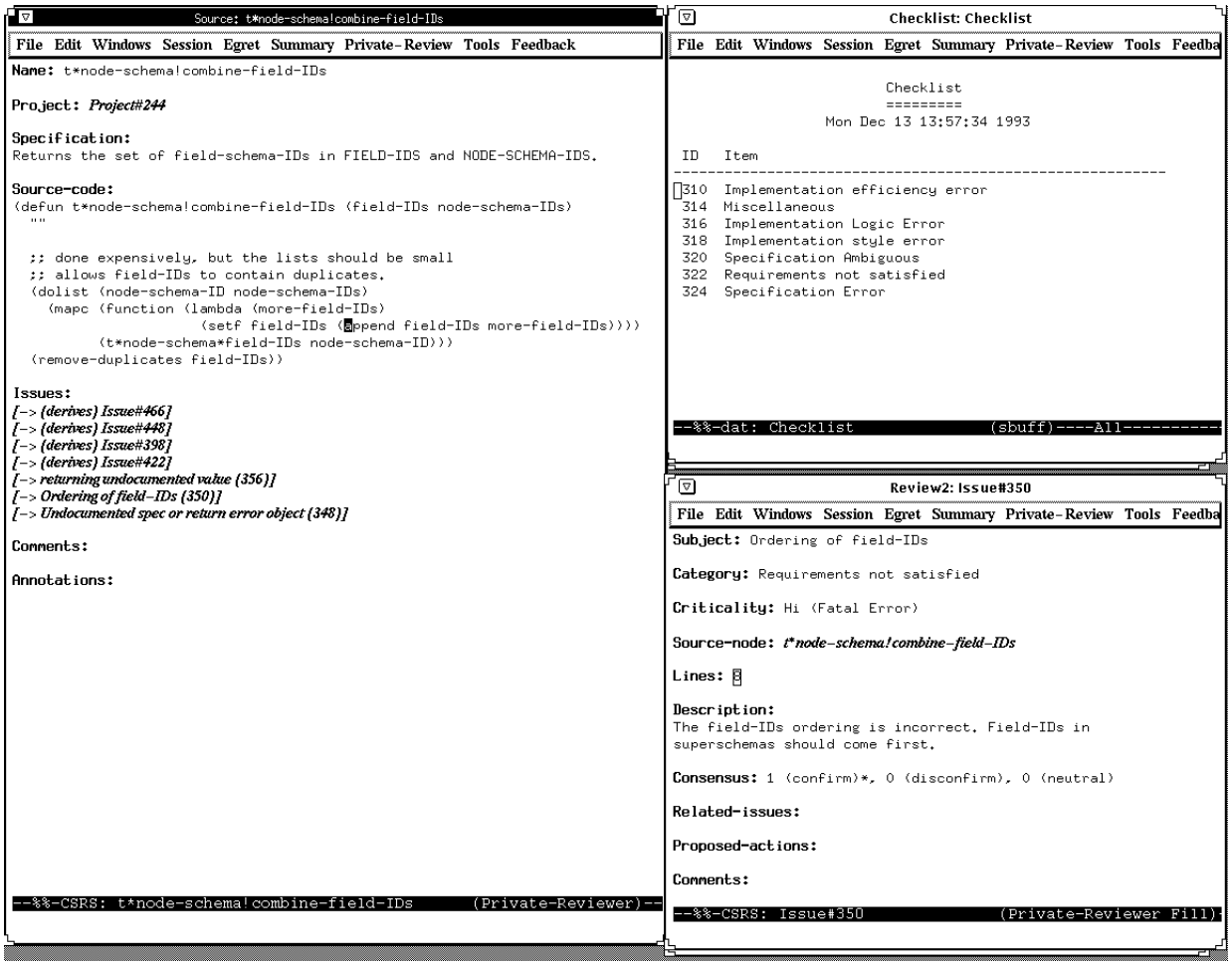

Figure 2: A CSRS screen illustrating the generation of an issue.

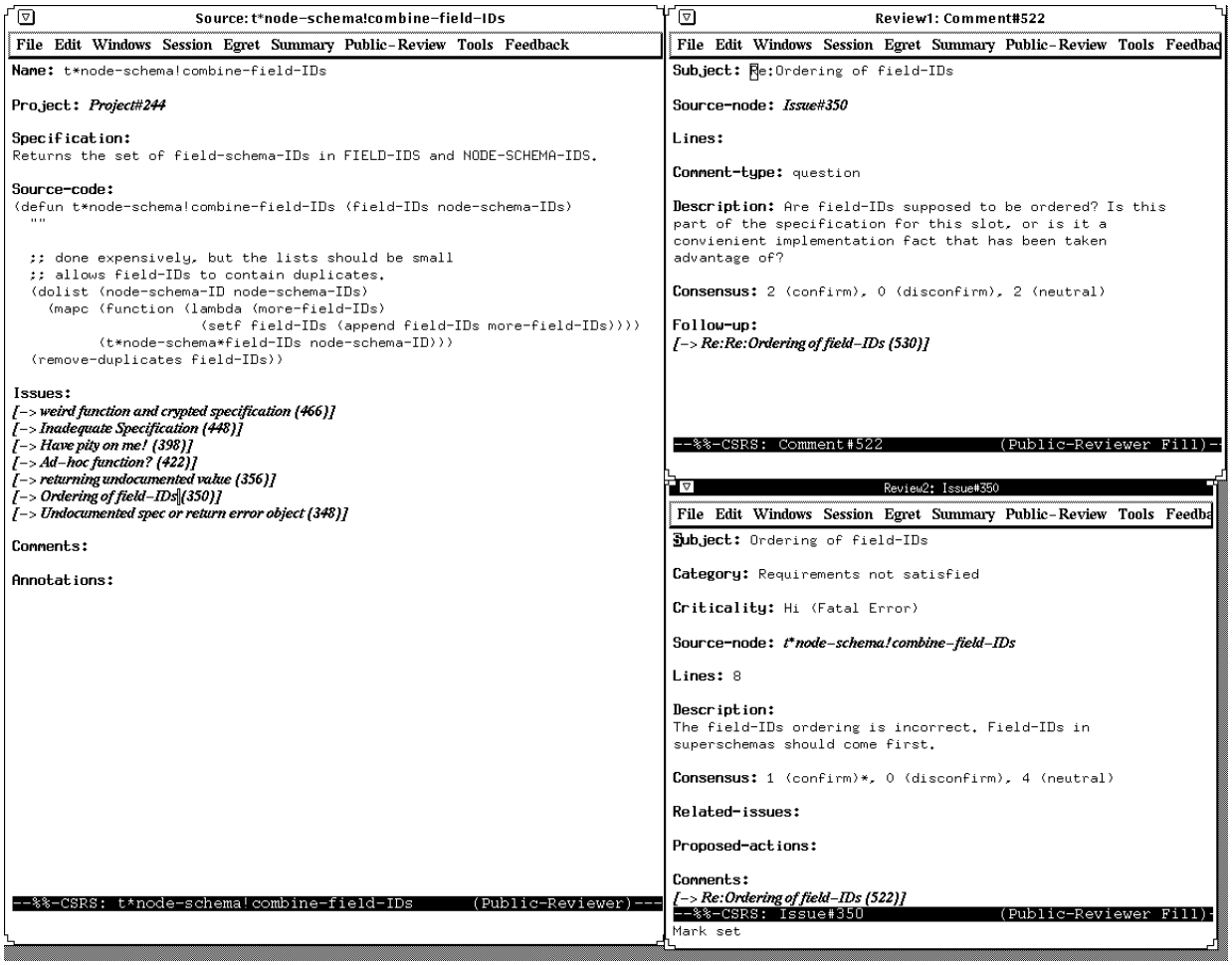

Figure 3: A CSRS screen from the public review phase.

severity, and implications of the issues raised during private review, as well as to address any additional issues that are discovered during this process. Reviewers analyze each of the issues raised, and vote by specifyng one of: confirm, disconfirm, or neutral. Any of these votes can be annotated with an explanatory comment.

Figure 3 shows a screen configuration from the public review phase of the Type-I review. This screen shows the same review artifact from Figure 2 on the left hand side, as well as the same issue in the lower right hand side. In this figure, however, that issue has been voted upon by other reviewers, and the upper right hand side displays one comment created in reaction to this issue. In fact, the "Follow-up" field to this comment contains a link anchor to the next node in this conversational thread.

As in public review, CSRS generates a log file containing a timestamped listing of the essential semantic actions taken by reviewers, along with idle-times. In addition, CSRS automatically generates a daily e-mail message to each reviewer if there are any new additions to the public debate that the reviewer has not yet seen.

After public review, the moderator uses CSRS to "boil down" the commentary generated by the reviewers into a Consolidated Review Report that concisely represents the status of review. If all

Issue 634: Ordering of field-IDs

Sources: t*node-schema!combine-field-IDs

Lines: 8

Consensus: 1 (confirm), 0 (disconfirm), 4 (neutral)

Description: The field-IDs ordering is incorrect. Field-IDs in superschemas should come first.

Discussion:
- Comment 522 (follow-up of Issue 350): Are field-IDs supposed to be ordered? Is this part of the specification for this slot, or is it a convienient implementation fact that has been taken advantage of?
- Comment 530 (follow-up of Comment 522): I guess this requirement is emerged due to node-buffer specification/implementation. The nbuff follows strict ordering of the field-IDs in a node instance. If you mesh up this ordering, you mesh up your nbuff.

Decision: This will be fixed. Also, the mapc form should be eliminated, since append is used to concatenate the two lists.

Figure 4: An example page from the CSRS final report for the Type-I review.

reviewers reached consensus about the issues raised, then the review terminates at this point, and the Consolidated Report becomes the final report of the review concerning the software artifact. CSRS provides tools to support the generation of a nicely formatted LaTeX document containing this report. Figure 4 shows one page from the Consolidated Report generated for the Type-I review. The Consolidated Report is not the only product of review, however. Other reports describing the measurements and analysis of the FTR process are also available, as discussed below.

For certain issues, consensus may not be reached through an on-line process, or their resolution may require higher-bandwidth than an on-line, asynchronous process. To handle these requirements, public review is optionally followed by a synchronous, face-to-face group review meeting. In this meeting, only those issues not resolved through the on-line debate are raised and resolved. The results from this meeting are entered into CSRS and a revised Consolidated Report is generated.

This description is intended only to provide a "gestalt" feel for CSRS, as context for the remainder of this paper, which focusses on the design of high quality measurement for formal technical review. While the preceding discussion presented CSRS from the reviewer's perspective, several other roles exist and are crucial to fully appreciating the system:

- The *administrator* is responsible for defining the precise CSRS data and process model to be used during review. For example, the administrator determines the specific artifact analysis technique, such as whether or not checklists will be used and what items will appear in the checklists. Administrative activities occur during the Process Improvement meta-phase, described in Appendix A.

- The *moderator* is responsible for overseeing the actual review and monitoring the on-line process, including transitioning between phases and producing the Consolidated Report.
- The *producer* is the person who created the artifact under review. The producer is responsible for assisting the moderator in setting up the hypertext network, and for answering certain types of questions raised during any phase of review.
- The *analyst* is responsible for reviewing the measurements collected during each review experience, comparing them to prior measurements, and generating hypotheses about how to improve review efficiency and effectiveness based upon this empirical data.

Of course, a single person may adopt more than one of these roles during a review.

CSRS is implemented as a client-server system, where a C++ database server process communicates with highly customized Lucid Emacs clients. CSRS is built on top of the EGRET collaborative work environment [13].

Our insights into FTR instrumentation have been gained through on-going review activities in our research group over the past two years. (Initially, each review experiment was followed by a fairly long period of redesign of CSRS. Currently, the system has stabilized and our group uses it regularly for quality assurance of the software developed in our lab.) Figure 5 summarizes some key data concerning our review experiments. The next sections discuss the insights we have gained from these experiments into measurement of FTR and design for instrumentation.

Review Experiments	9
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 5: *Ranges of key review statistics.*

3 High Quality Measurement in CSRS

3.1 Coarse-grained measurement

Measurement of traditional manual review practice results in coarse-grained statistics concerning the *outcome* of review, such as the number of defects found, their breakdown with respect to the type and severity of defect, their relationship to the size of the artifact under review, and so forth. Manual review practice also results in coarse-grained statistics concerning the *process* of review, such as reviewer-supplied estimates of the time spent preparing for review and participating in the meeting.

Unfortunately, high quality collection of even these coarse statistics can be difficult and expensive. Weller's report on 6000 manual review experiences at Bull HN states that the preferred method involved

the transcription by clerical staff of paper-based review commentary into a PC-based record-keeping system, even though this introduced errors and led to the need for various "data sanity" checks by the technical staff[4]. A report on Digital's internal educational practice reveals that even after 10 years of experience with a software inspection curriculum, they were still unable to consistently train technical staff to effectively collect and utilize data on the review process [12]. As a final example, one respondent to our informal USENET survey on FTR noted that his organization had been collecting review data for over 10 years, but had never actually used any of it due to the overhead of manual collection, entry, and analysis.

By moving the review process on-line, CSRS trivially resolves some of the traditional difficulties and expenses associated with the collection of coarse-grained outcome statistics. For example, Weller states that one of the two major sources of review measurement error in his organization is module size, presumably because this value is calculated by hand by clerical staff. In an on-line system such as CSRS, however, the size of each review artifact is calculated automatically and reliably. Similarly, outcome-oriented data on the number of defects found, their breakdown with respect to type and severity, and the location of the source artifact in which they appear are reported by CSRS automatically and reliably in the Consolidated Report.

CSRS also provides coarse-grained process statistics concerning effort similar to those in traditional manual review. Once again, reliable collection of coarse-grained process statistics is problematic in manual review methods. For example, inadequate reviewer preparation for the group meeting is a substantial problem in traditional review [6]. Rather than simply confess to insufficient preparation, a reviewer may instead try to "bluff" their way through the meeting, exaggerating their preparation time and then generating comments "on the fly". Inadequate preparation can fundamentally undermine the entire review process and measurement interpretation, since it is difficult to distinguish the review of a high quality artifact by well-prepared reviewers (that generates a small number of minor issues) from the review of a moderate or low quality artifact by ill-prepared reviewers (that also generates a small number of minor issues.) In traditional review practice, draconian responses are suggested to detect insufficient preparation, such as having the moderator omit one page from the review materials, and checking to see that all reviewers request the missing page before the group meeting begins [6].

Once again, moving the review process into an instrumented, on-line environment can alleviate these difficulties. CSRS keeps track of the effort spent by each reviewer during the analysis of the review artifact. This information about each reviewer is also available to the moderator (although reviewers do not have access to each other's process statistics). Thus, there is no ambiguity about the preparation effort, and while reviewers may not always adequately prepare, at least this fact will be visible and can be noted as a qualification to the review outcome. Additional process support, such as quality assurance tests that must be explicitly checked off by the reviewers, can be used to increase the rigor and confidence in the review.

Figure 6 shows a table containing coarse-grained outcome and process statistics from the Type-I review. This table can be generated automatically by CSRS without any clerical transcription and without the need for data sanity checks by the technical staff.

3.2 Fine-grained measurement

The fact that CSRS simultaneously increases the quality of coarse-grained statistics about review process and outcome and decreases the overhead of their collection is a useful contribution. CSRS

[4] This method was preferred even over direct entry by technical staff into a notebook PC during meetings.

Coarse-grained statistics for Type-I Review	
Review Start Date:	27-Sep-93
Review End Date:	13-Oct-93
Artifact Size (LOC):	557
Number reviewers:	6
Total issues/comments:	76
Private review time (person-hours):	9.7
Public review time (person-hours):	9.0
Group review time (person-hours):	4.5

Figure 6: High quality, coarse-grained review measures.

User-ID	Operation	Node-ID	Name	Date	Time
68	connect		CSRS	9/29/93	17:10:03
68	summarize		Summary-sources	9/29/93	17:10:18
68	read-node	276	t*nschema	9/29/93	17:10:36
68	set-status	276	t*nschema	9/29/93	17:10:42
68	close-node	276	t*nschema	9/29/93	17:10:44

Figure 7: An example event log from the first minute of one review session.

goes even farther than this by providing high quality collection of *fine-grained* statistics about review process and outcome, again without incurring review overhead.

Fine-grained outcome statistics include the number of minutes spent by each reviewer on each hypertext node in the review artifact. The Source-nodes Summary window in Figure 1 shows one place in which this data is available in an on-line, cumulative fashion; CSRS can also generates spreadsheet data files containing this information. Fine-grained process statistics include the precise sequence of actions taken by reviewers to perform the analysis, including how the review artifact was initially read, how many sessions of what duration were required, when and how issues were generated, and so forth. Fine-grained statistics about outcome and process provide a qualitatively different level of insight that opens the way to new forms of inquiry into formal technical review.

The ability of CSRS to collect fine-grained measures is a result of a combination of design decisions. First, CSRS re-represents all review artifacts as a hypertext document, where each node contains a single semantic "unit" of the artifact at a fine grain size. As discussed in Section 5.2, other computer-supported review environments represent review artifacts at a coarser grain size, such as the file or document level, or do not exploit the use of hypertext for instrumentation.

Second, CSRS monitors each reviewer as they use the system, and unobtrusively generates a log file of timestamped "CSRS events" in background during each session. This log file is cached locally at the client process during the session, then written out to the database server process at disconnect time. CSRS provides a variety of tools for post-processing of this raw instrumentation data.

Given that each node appears in its own window, CSRS can detect the focus of reviewer attention by generating events to record display and dismissal of each CSRS node during the review session. CSRS also records the occurrence of interesting reviewer actions, such as voting or marking a node as reviewed, that do not result in window activity.

Figure 7 shows a small portion of the timestamped event log for the Type-I review. (Over 4300 timestamps were generated during this review.) It is easy to see that certain measures, such as the

elapsed time a node is reviewed, could be calculated by simply subtracting the time that a node is closed from the time the node is read from the server database process.

Although this approach appears reasonable at first glance, our initial experiences (in reviews prior to Type-I) quickly demonstrated otherwise. In a multi-tasking, multi-windowing system such as Unix and X-windows, reviewers may interrupt their CSRS activities to read their e-mail or perform other computer-related functions. Perhaps the phone rings, or a colleague interrupts with unrelated business. When dedicated workstations are involved, a reviewer might leave the CSRS system up and running over the weekend in the middle of review, as we discovered to our chagrin when we measured an apparent effort of over 50 hours by one reviewer for a small 5 line function!

To account for each of these types of idle time, we augmented the "top down" timestamping with a timer-based, "bottom-up" timestamp process that wakes up once per minute, determines whether there has been any low-level editing activity (such as a keypress or mouse movement) within CSRS during the past minute, and if so, generates a "busy" timestamp in the event log.

This combination of top-down and bottom-up timestamping has proven to be very effective: it provides us with a record of the "interesting" user activities from a measurement perspective, as well as providing an accurate reflection of the time spent by the reviewer actually working with CSRS, with a precision of plus or minus one minute.

We have investigated one potential flaw in this approach: what about a reviewer who simply "stares at the screen" for many minutes at a time? CSRS would represent such behavior as idle time, even though the reviewer was hard at work. Fortunately, empirical analysis of our timestamp logs indicates that such a behavior rarely, if ever, happens, at least during our laboratory studies. We investigated this by calculating the inter-event interval for top-down events and found that timestamps are typically generated less than 30 seconds apart over 80% of the time, less than a minute apart over 90% of the time, and less than three minutes apart over 98% of the time.

Once again, the hypertext structure of the review artifact plays an essential role in high quality data collection. Since the artifact is presented to the reviewer in individual small nodes, it appears to be very rare that a user will spend many minutes on a single node without *any* keyboard activity (including scrolling the screen, moving the mouse to "keep your place", etc.)

4 Applications of High Quality Measurement

4.1 Improving coarse-grained measurement applications

High quality, fine grain measures of FTR provided by CSRS improve traditional empirical inquiries into FTR-based software quality improvement, as well as support entirely new forms of empirical investigation.

FTR measures are commonly used to provide insight into review productivity, such as the number of errors found per unit size of an artifact, and the cost to detect them. Researchers suggest that such a cost-effectiveness metric can be used as a driver for changes to the FTR process, such as increasing or decreasing the number of reviewers, shortening or lengthening the review time, changing the specific analysis technique used for review, and so forth.

While feasible in principle, such empirically-based process improvement is very difficult to practice with traditional, manual methods. First, as noted above, collection and analysis of FTR data is very expensive and requires significant quality assurance[5]. Thus, an organization must be willing to

[5] In other words, the quality assurance data requires quality assurance :-)

commit new resources over and above the basic resources for FTR. In effect, management is betting that this additional investment will result in data that can be used to improve the quality and efficiency enough to offset this investment. This may explain why only very large corporations with a strong commitment to process improvement have published findings on FTR measurements, and why our informal USENET survey finds such low percentages of FTR data usage.

Second, in order to provide a scientifically sound basis for process change, the organization must be willing to undertake FTR under controlled experimental conditions in order to minimize the effect of extraneous procedural variables on the outcome data. In addition, the organization must replicate the FTRs enough to yield statistically significant conclusions that warrant changes to the FTR process. Once again, only very large corporations have the time and sufficient numbers of FTRs required to perform such empirically-justified process improvement. FTR data published by different organizations is virtually incomparable, since these studies do not specify the technique or control its application sufficiently to support experimental replication within another organization.

CSRS addresses both of these issues. First, collection of FTR data in CSRS is automated, and CSRS provides analysis tools to automate the generation of many popular measures of cost-effectiveness from the raw data. Thus, an organization needs commit only the resources for formal technical review; no additional resources are required for measurement collection and initial analysis.

Second, CSRS intrinsically controls for many of the extraneous variables that can weaken the validity and comparability of manual FTR measurements. Since the review process is on-line, and since CSRS can enforce a particular process and analysis technique, review experiment designers can both specify a particular style of review, and analyze the data to see if it was actually followed. As a result, CSRS data is much more amenable to cross-organizational comparison than any manual technique. This means that smaller organizations with less resources can both contribute to and profit from the experiences of others with CSRS. (As CSRS technology is transferred into industry, we intend to set up an on-line public database of CSRS metric data to support such inter-organizational cooperation.)

4.2 Fine-grained measurement applications

While CSRS can provide significant improvement to traditional coarse-grained measurements of FTR, perhaps the most exciting aspect of its design is that it enables entirely new forms of inquiry into review processes through fine-grained instrumentation. In this section, we will overview two ongoing research studies that exploit this capability.

4.2.1 Complexity/Effort Relationships

The overall effort required for formal technical review is difficult to predict at other than very general levels, such as “15% of the entire software development effort”. This information is useful from a historical perspective, but provides little help to allocating specific resources to specific review artifacts. The need for better estimators of review effort has long been identified as an important open problem in review practice [5].

The fine-grained instrumentation in CSRS provides a novel approach to this problem. Since both the review artifact and the precise time and effort spent upon each component of it by each reviewer are captured by the system, it may be possible to correlate features of the review artifact to effort expended and generate a predictive measure of effort. A sufficiently precise correlation would allow

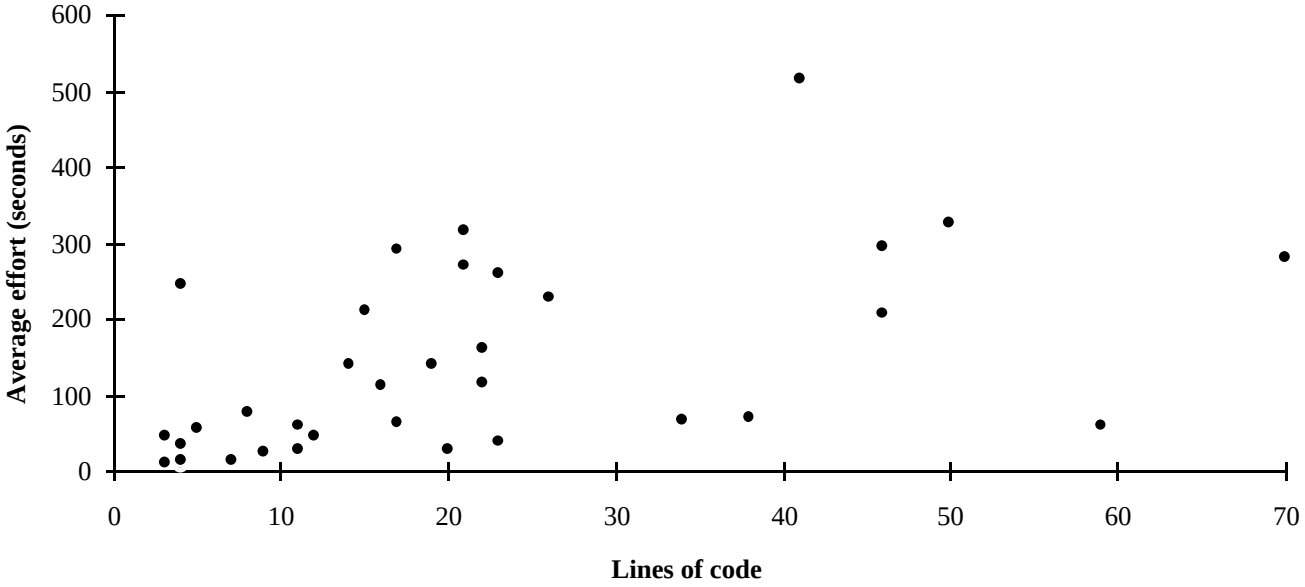

Figure 8: A scatter plot of review effort vs. one size-based complexity measure: lines of code.

an organization to predict the resources required for review activities based upon the complexity computed for the code, or conversely, to fit the set of artifacts to the resources available for review.

We are currently studying this possibility for the review of code artifacts. In this study, we are investigating the relationship of review effort to three size-based measures of software complexity: Halstead's volume measure [10], McCabe's cyclomatic number [19], and lines of code.

Figure 8 illustrates some of the data we have collected so far. There is no statistically significant *linear* correlation between effort and any of the three measures of complexity for this data set, although, with some imagination, a general positive correlation might be seen. Thus far, lines of code is the most predictive of the three complexity measures. The most important result of these initial analyses has been a refinement of our experimental hypotheses. We will continue to add new review experiments to our data set to see if, given an increased sample size, a statistically significant trend emerges. In addition, these analyses have suggested to us that other factors, such as reviewer experience, may be an important influence on review effort. We will be investigating these alternative hypotheses as well, which may result in a multi-factored predictive function of review effort, similar in spirit to the COCOMO model.

4.2.2 Protocol Analysis of Reviewer Behavior

If formal technical review methods are to reach their full potential, much more needs to be known about the details of both individual and group FTR behavior. Past research on FTR has determined that the effectiveness of FTR will vary tremendously depending upon the review technique employed and the quality of the group process. Yet little is known about the precise "styles" of review adopted by different people, the structure and process of consensus building during review, and the effect of these behaviors upon the review outcome. Precise measurement of the process of formal technical review is the first step toward understanding, and ultimately, improvement.

The design of CSRS provides unique insight into the fine-grained process of individual and group

review. By breaking down a review artifact into its constituent components, representing it as an interlinked hypertext network, and recording the patterns of traversal and actions of the reviewers, CSRS provides a qualitatively different level of process representation than is possible in other manual or automated approaches to review.

This representation does, however, require automated support tools, since many thousands of individual timestamped review events are recorded during a typical review experience. We have implemented a visualization tool for review called Timeplot to support our initial inquiries in this area. Timeplot processes the review event logs, and automatically generates a LaTeX document illustrating the contents of each CSRS screen during each minute of review. Timeplot also reveals patterns of idle time during a review session. Figure 9 shows the process of one review session generated by Timeplot for the exemplary review experience.

We are currently engaged in analyzing this fine-grained behavior from a variety of perspectives. We are working on a grammar-based approach to extracting patterns from the sequence of reviewer actions, with the goal of identifying typical behaviors across reviewers, and to identify higher-level activities occurring during the review process. Such work builds upon recent research in protocol analysis [22] and statistical techniques [9].

5 Related Work

Research related to CSRS falls into two general categories: research on formal technical review methods, and research on computer-supported FTR.

5.1 Formal Technical Review Methods

Code Inspection. The seminal formal technical review method is Fagan's code inspection [5]. It begins with a presentation by the producer about the material to be reviewed. Next, reviewers prepare for the review meeting by informally analyzing the review artifact. The central and most precisely prescribed activity is the inspection meeting, which is a face-to-face group examination and issue consolidation activity. The reader paraphrases the source materials, statement by statement, and the participants interrupt with questions that may eventually lead to discovery of errors. Errors are recorded and fed back into future review sessions. Fagan's code inspection method presumes *no* automated support technology.

Software Inspection. Gilb's Inspection method [7] is a direct descendent of Fagan's Code Inspection. However, Gilb introduces a substantial number of refinements and elaborations to Code Inspection, and generalizes it for all artifacts produced during software development. Also similar to Fagan, Gilb's method is entirely manual in nature.

Software Review. Humphrey's software review [11] begins with group comprehension meeting where participants review background materials presented by the producer. Next, reviewers analyze the review artifacts individually, using a error checklist to guide their analysis. The producer then correlates and consolidates the errors found by individual reviewers. The final phase is a group meeting where the producer presents the error list item by item, and the participants discuss the significance of the errors and decide upon an action.

Session 376: Sep 29, 1993 17:10:03

```
Phase: Private

Real  Active   Source                          Review2       Other Events
Time  Time     Screen                          Screen

17:10 00:00    -
  :     :      |t*nschema                                    Reviewed t*nschema.
  :     :      |t*def-set-attribute
17:11 00:01    -
  :     :      |t*def-add-attribute-vals                     Reviewed t*def-add-attribute
  :     :      |t*def-set-attribute
17:12 00:02    -
  :     :      |t*def-delete-schema                          Reviewed t*def-set-attribute
  :     :      |
17:13 00:03    |
  :     :      |
  :     :      |
17:14 00:04    |
  :     :      |
  :     :      |
17:15 00:05    |
  :     :      |
  :     :      |
17:16 00:06    |
  :     :      |
  :     :      |
17:17 00:07    |
  :     :      |
  :     :      |
17:18 00:08    |                               -
  :     :      |                               |Comment#374  Confirmed Comment#374.
  :     :      |                               |
17:19 00:09    |                               |
  :     :      |                               |
  :     :      |                               |
17:20 00:10    |                               |
  :     :      |                               |
  :     :      |                               |
17:21 00:11    -                               -
```

Figure 9: An excerpt from an example Timeplot graph from one review session during the Type-I review. This session actually lasted for 22 minutes, but only the first 10 minutes are shown here.

Active Design Review. Active Design Review [20] is conducted in four phases. First, the producer presents a brief overview of the module under review. Next, reviewers analyze the module individually. Active Design Review specifies a novel method for this individual review, where each reviewer must provide answers to a set of questionnaires designed by the producer, thus guaranteeing comprehension. Furthermore, individual reviewers are utilized as specialists, and review only sections of the artifact. Next, the producer meets with the reviewers individually to discuss the concerns raised in the questionnaire. Finally, the producer creates a consolidated report.

Cleanroom Inspection. Cleanroom Inspection is a component of Cleanroom software development [4]. In general, Cleanroom Inspection consists of individual review, followed by consolidation by the review leader. The novel aspect of Cleanroom Inspection is that analysis of the review artifact involves step-wise verification of a formal specification against the actual source code.

Phased Inspection. Phased Inspection [17] involves two kinds of review phases—single inspector and multiple inspector. Single inspector phases involve mostly simple clerical procedures to establish basic structural or syntactic properties of the review artifact, and that require only minimal comprehension of the artifact. After all single inspector phases complete, the multiple inspector phases begin. These phases involve active comprehension and group-based analysis of the review artifact, and may involve both standard checklists (such as in Fagan Inspection) and active checklists (such as in Active Design Review). The final phase is reconciliation, in which the the inspectors compare their findings in a group meeting using Delphi-like technique. Phased Inspection, like CSRS, is predicated upon computer support, and its system, called INSPEQ, is discussed briefly in the next section.

5.2 Computer-supported FTR systems

ICICLE. ICICLE [3] is an X window-based system that was designed to support Fagan's Code Inspection on C language software. ICICLE allows reviewers to perform the code inspection meeting on-line, providing various tools to help control the paraphrasing process and generation of issues. The ICICLE experience reveals some of the difficulties encountered when trying to "faithfully reproduce" an intrinsically manual process in a computer-mediated environment. ICICLE did not include instrumentation support for other than simple statistics on the total number of defects discovered and their severity. ICICLE was oriented only to C code, and represented artifacts at the granularity of files.

INSPEQ. INSPEQ [17] is an X window-based support environment for the Phased Inspection review method. It provides a multi-window based environment to support display of source code, checklists, standards, and comments. INSPEQ represents source artifacts as files, and provides some coarse-grained instrumentation to record elapsed time and the numbers of issues raised.

Scrutiny. Scrutiny [8] is an X window-based system developed at Bull HN Information Systems originally to support inspection-at-a-distance, following Fagan's method. Unlike other on-line face-to-face systems such as ICICLE, Scrutiny can be applied to a wide range of documents, and is built on top of ConversationBuilder, a general purpose, collaborative support environment [16]. Scrutiny inherits the hypertext capabilities of ConversationBuilder, but has no representation of time, thus severely limiting its instrumentation capabilities.

CSI. CSI [18] is a system developed at the University of Minnesota to allow inspection-at-a-distance using Humphrey's method. CSI uses an X window-based environment called Suite that provides hypertext capabilities, and supports both asynchronous and synchronous meeting capabilities. CSI does support certain coarse-grained time-based measures.

6 Design for Instrumentation

In conclusion, perhaps the most important lesson learned from our experiences with CSRS is this: one must design with instrumentation explicitly in mind, if one is to gather useful empirical data from a computer-mediated process. For example, while the use of hypertext provides certain representational benefits, it adds most value by revealing and tracking the user's focus of attention. In future, we hope to extend our instrumentation for this crucial measure through the use of eye-tracking devices. This instrumentation will reveal which individual symbol the user fixates upon with a precision of milliseconds. This will provide an entirely new, qualitatively different representation of reviewer activity, and enable correspondingly new experimentation and insight into the review analysis process.

Our design experiences have also convinced us that traditional FTR methods, such as Fagan's Code Inspection, are designed for and inextricably bound to a non-computer supported context. As formal technical review becomes increasingly computer-mediated, it is essential to devise new methods that exploit the strengths and minimize the weaknesses of this entirely different context for collaboration. For example, virtually all manual review methods embrace the edict to "raise issues, don't resolve them." We believe that this may only be appropriate within a synchronous, face-to-face group process. In FTArm, entwining issue generation, discussion, and resolution is natural and appears to provide significant advantages.

On the other hand, our experiences have also convinced us that while FTArm is a powerful method, it has weaknesses as well as strengths. For example, its asynchronous nature is helpful in reducing the cost intrinsic to synchronous meetings, but some organizations may require more "synchronicity" in their reviews than FTArm provides. FTArm is also a complex method that is not well-suited to initial adoption of computer-supported FTR by an organization. CSRS includes an FTR method definition language so that organizations can design methods with tradeoffs appropriate to their own context.

By eliminating the cost of FTR measurement, CSRS can serve as infrastructure for a new paradigm of research and collaboration in the software engineering community. We are currently negotiating technology transfer arrangements with several industry organizations. A goal of this process is to validate our design and instrumentation within several organizational contexts, and to learn the requirements for unaided adoption of this technology. If you are interested in participating in our research on computer-supported formal technical review, please visit our World Wide Web home page at "http://www.ics.hawaii.edu/~csdl/csrs".

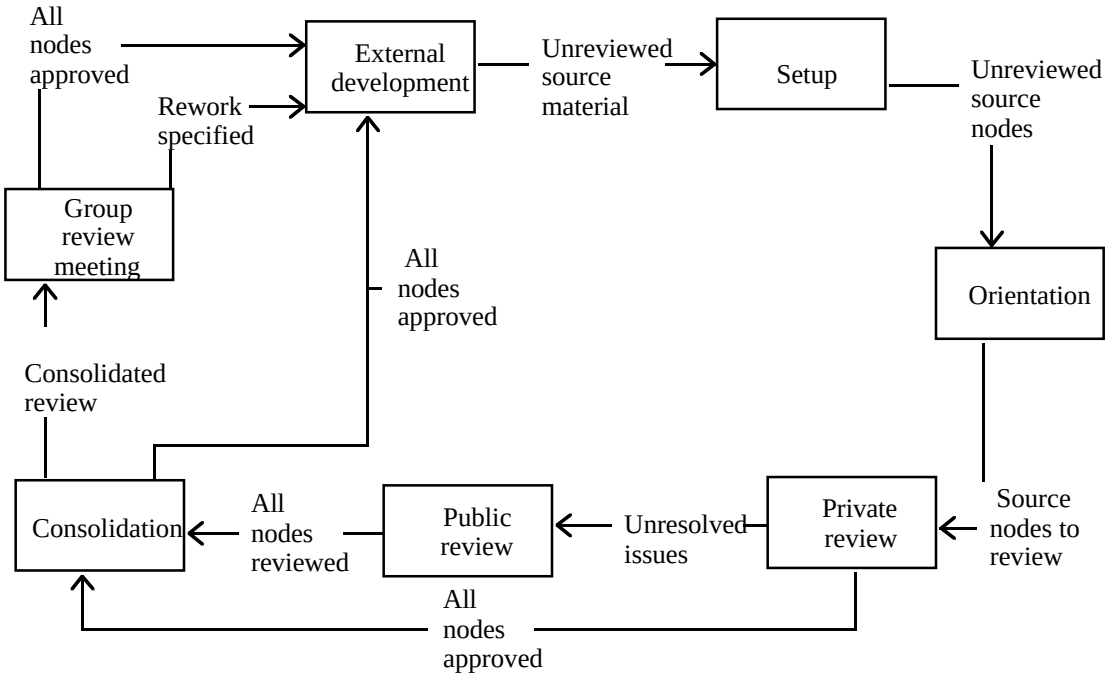

Figure 10: The CSRS process model.

A The FTArm Method

The FTArm method for CSRS involves seven well-defined phases, as illustrated by the diagram in Figure 10. It also involves a *meta-phase* for FTR process improvement as a result of measurements analysis. An informal description of each of the seven phases and the meta-phase are provided below.

Setup Phase. In general, each FTArm review begins with the downloading of the software review artifacts, such as requirements specifications, designs, source code, test plans, and so forth from their ASCII files into the CSRS database. CSRS semi-automatically re-structures the artifacts as a fine-grained hypertext network consisting of typed nodes and links. For example, for source code artifacts, each function and variable definition would be stored within its own node, with hypertext links to related nodes. For a requirements specification, CSRS would re-structure the document hierarchically into a set of nodes for each major section, each containing subnodes for individual specifications typed according to their focus. During the Setup Phase, the moderator and/or the producer also decide upon the composition of the review team and the artifacts to be reviewed.

Orientation Phase. This phase prepares the participants for private review by introducing them to the artifacts under review. The exact nature of this phase depends upon the kind of review and the review group. It may range from a formal overview meeting with a presentation by the producer about the review artifact structure and behavior, to an informal notification through e-mail noting the presence of new artifacts to review. The Orientation phase assumes that the participants are familiar with CSRS; if not, a separate CSRS training session must be provided.

Private Review Phase. In this phase, reviewers inspect artifacts privately and create issue, action and/or comment nodes. Issue and action nodes are not publicly available to other reviewers at this time, but comment nodes are publicly available. Comment nodes allow reviewers to request clarification about the review artifact, or review process, and may also contain answers to these questions by other participants. CSRS supports three common techniques for analysis of review artifacts: free review, checklist-based review, and verification-based review.

Public Review Phase. In this phase, all nodes are made public, and all review participants (including the producer) react to the issues and actions by voting and creating new nodes. Participants can create new issue, action or comment nodes based upon existing nodes. Voting is used to determine the degree of agreement within the group about the validity and implications of issues and actions. This phase normally concludes when all nodes have been read by all reviewers, and when voting has stabilized on all issues. During both public and private review, CSRS automatically sends a daily e-mail message to any participants who have nodes for review or voting.

Consolidation Phase. In this phase, the moderator analyzes the results of public and private review, and produces a condensed written report of the review thus far. In contrast to traditional FTR reports, which typically contain only a checklist of raised issues 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.

Group Review Phase. If the consolidated report identifies issues or actions that were not successfully resolved via public and private review, a group meeting is the next step. In the meeting, the moderator presents 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 hardcopy document representing the product of review. CSRS-basd support for this face-to-face meeting is currently under way.

External Development Phase. During this phase the software is enhanced or corrected in response to the issues raised during review or due to other development processes.

Process Improvement Meta-Phase. The seven phases above provide a framework for the FTR process, but also allow for evolution in response to the measurements supplied by CSRS. Many review factors can and should be methodically tested within the CSRS framework to discover their optimum values within a particular organization and application context. Some of these factors include: 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.

B Bibliography

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

[2] 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.

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

[4] M. Dyer. Verification-based inspection. *Proceedings of the Hawaii International Conference on System Sciences*, 2:418–427, 1992.

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

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

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

[8] 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.

[9] J. M. Gottman and A. K. Roy. *Sequential analysis: A guide for behavioral researchers*. New York: Cambridge University Press, 1990.

[10] M. Halstead. *Elements of Software Science*. North-Holland, 1977.

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

[12] Tony Hutchings, Michael G. Hyde, David Marca, and Lou Cohen. Process improvement that lasts: an integrated training and consulting method. *Communications of the ACM*, October 1993.

[13] Philip M. Johnson. Experiences with EGRET: An exploratory group work environment. *Collaborative Computing*, 1(1), 1994.

[14] 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.

[15] Philip M. Johnson. Supporting technology transfer of formal technical review through a computer supported collaborative review system. In *Proceedings of the Fourth International Conference on Software Quality*, Reston, VA., October 1994.

[16] Simon M. Kaplan, William J. Tolone, Douglas P. Bogia, and Celsina Bignoli. Flexible, active support for collaborative work with ConversationBuilder. In *Proceedings of the ACM 1992 Conference on Computer-supported Cooperative Work*, 1992.

[17] John C. Knight and E. Ann Myers. An improved inspection technique. *Communications of the ACM*, 11(11):51–61, November 1993.

[18] Vahid Mashayekhi, Janet Drake, Wei-Tek Tsai, and John Riedl. Distributed, collaborative software inspection. *IEEE Software*, 10(5), September 1993.

[19] T.J. McCabe. A software complexity measure. *IEEE Transactions on Software Engineering*, 2(6):308–320, December 1976.

[20] 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.

[21] M.C. Paulk. Capability maturity model for software, version 1.1. CMU/SEI-90-TR-25, Software Engineering Institute, 1993.

[22] John Smith, Dana Smith, and Eileen Kupstas. Automated protocol analysis: Tools and methodology. Technical Report TR91-034, University of North Carolina at Chapel Hill, Department of Computer Science, 1991.

[23] Edward F. Weller. Lessons learned from three years of inspection data. *IEEE Software*, pages 38–45, September 1993.