Anywhere, Anytime Code Inspections: Using the Web to Remove Inspection Bottlenecks in Large-Scale Software Development

or

Studies in Interval Reduction in Large-Scale Software Development

James Perpich, Dewayne E. Perry, Adam Porter, Lawrence G. Votta and Michael W. Wade

ABSTRACT

Disseminating information, maintaining artifact consistency, and scheduling coordinated activities are critical problems in any large-scale, software development. Inadequate management of this "process overhead" can increase rework effort, decrease quality, and lengthen interval. These problems are greatly magnified when a development team is divided across two or more geographcally separate locations.

For example, in traditional development settings, conflicts in scheduling meetings account for a significant portion of inspection interval [illegible] . In a distributed development, inspection interval is lengthened still more by delays resulting from time-zone mismatches, travel to meetings, and long-distance (sometimes international) mailings.

In this article we present a tool, Hypercode, that supports meetingless software inspections with geographically-distributed reviewers. HyperCode is a platform independent tool, developed on top of an internet browser, that integrates seamlessly into the current development process. By seamless we mean the tool produces a paper flow that is almost identical to the current inspection process, and is consistent with ISO certification. Furthermore, HyperCode's user acceptance has been excellent.

More importantly we evaluated and compared the cost-effectiveness of HyperCode inspections with that of manual inspections. We found that the cost savings from reduced paper work and the time savings from faster distribution of the inspection package have been substantial. These savings together with the seamless integration into the existing process appear to be the major reasons for tools acceptance.

From our viewpoint as experimentalists, however, this acceptance came too readily and too easily: our control group insisted on using HyperCode. Therefore, we were unable to directly assess HyperCode's impact of inspection quality. Nevertheless, by using historical data we can show that meetingless inspections (like those supported ny HyperCode) are at least as effective as traditional inspection with meetings.

Keywords

Code inspections: web-based, meetingless, asynchronous; Natural occurring inspection experiment; Automated support for inspections; Work, paper and information flow.

INTRODUCTION AND BACKGROUND

Code inspections have become a standard part of effective software development processes. Indeed, code inspections have been so successsful that virtually all the major artifacts in the software life-cycle are subjected to some form of group inspection. Because inspections are a common and frequently recurring activity, they are likely to be affected by changes in the contexts surounding software development. Moreover, they are also prime candidates for improvements, just as any frequently executed piece of code in the systems we build is also a candidate for such scrutiny.

Large software projects have always suffered to some extent from geographical separation. People are often spread over multiple floors within the same building as well as distributed across various buildings and locations. The trend towards the globalization of software intensive products and their production intensifies this problem. This trend puts stress on our processes but also provides us with opportunities for process improvement. It is in this context that we explore the possibilities for reductions in both cost and interval while maintaining the requisite level of quality.

- - say what we did: had a goal, built a tool, look at alternatives, deployed it and think it was a useful pi and will discuss the process of buildingit, how we arrived a building it, and how well it has worked.

Our Process Improvement Approach

A typical process improvement approach is to introduce a process change (often involving a new tool), and then to evaluate the effect of that change. Although it is certainly necessary to evaluate process changes, too often the change is proposed without thoroughly understanding the existing process, its important problems, or the and range of alternative solutions and their tradeoffs.

Therefore, our approach to process improvement involves the following steps.

- **Understand the current process.** The starting point of any process improvement activity should be an understanding of the process. Perry, et al. [?] describe some techniques for conducting appropriate studies.
- **Identify key problems.** Once the process is understood, problem areas can be identified and prioritized according to an organzation's goals. For example, Bradac, et al. [?] combined process modeling and time usage data to identify and prioritize process bottlenecks. In another study, Perry and Stieg [?] collected data on the faults found in evolving a large, real-time system to identify classes of development problems and to prioritize them by frequency of occurrence and fix effort.
- **Explore and evaluate alternative improvements.** Potential improvements will have different strengths and weakneses and empirical studies are fundamental to determining them. These studies explore key issues, risks, and costs of alternative improvements, and may involve controlled experiments, surveys, process modeling, and prototype development and evaluation. For instance, Porter et al. [?] compare and evaluate alternative detection methods for software requirements' inspections. Jangadeesan et al. [] formally modelled an industrial software component, and implemented it in several different ways in order to evaluate a prototype test toolset,
- **Build and evaluate preferred improvement.** Based on the previous analyses one or more preferred improvements will be selected. In many cases, the preliminary evaluation will not be sufficient to determine the actual range of the improvement. In these cases, the improvements must be built and deployed before they can be properly evaluated. Again, empirical studies are one of our basic tools for their evaluation.

In the remainder of this paper we follow these steps to improve the code inspection process. Afterwards we discuss critical open questions and present our conclusions.

UNDERSTANDING THE ORIGINAL INSPECTION PROCESSES

The inspection process, abstractly, is divided into three basic phases: preparation, collection and repair. The preparation phases includes such things as initiating the inspection process, disseminating the inspection package, and the inspectors preparing (that is, inspecting the artifact) for the collection phase. The collection phase includes the collection, assessment and resolution of defects. The agreed upon defects are than fixed in the repair phase.

We first describe the original process in detail and then present some quantitative data about some critical apsects of this process.

The Process Description

For ease of comparison with the improved process discussed below, we present the original inspection process as a sequence of basic steps.

1. Modification Requests (MR's) are issued whenever additions or enhancements to code are needed.
2. A developer accepts one or more MR's and develops the necessary code.
3. The author then makes a code unit available for inspection. A code unit may implement one or more MR's.
4. The author selects his or her review team.
5. The author contacts the review team and schedules the inspection meeting. He or she coordinates the proposed schedule with project management.
6. The author prepares the inspection package and distributes paper copies of it to the review team. The inspection package includes the code unit' source text, information about meeting time and location, and all required forms.
7. Prior to the meeting, the reviewers analyze the code unit looking for defects.
8. The author and reviewers conduct the collection meeting. One of the reviewers is assigned to be the moderator, who makes sure the meeting does not get bogged down on any single point of discussion.
9. During the meeting the author creates the consolidated list of issues. Issues are the potential defects discovered during the inspection.
10. The author determines which issues must be repaired, and does so.
11. The author brings the reworked code to the inspection moderator who ensures that all issues have been addressed and signs off the inspection.

The original process automates much of the step 6: the code and the changes generated by the MRs are atomatically generated for printing and then manually distributed. The MR and design documents are made available for the reviewers but are not distributed as part of the package to save on the amount of paper generated.

Quantitative Data

- recorded vs analysis (paper vs on-line) data - ISO compliance

- data from live, large scale project - 150 diff things about inspection

- - - inspector characteristics (ype, expertise, etc), fault, time, etc

- - - avg fault density - 1 fault in 30

- desk vs meetings data

IDENTIFYING KEY PROBLEMS

Process Improvement Goals

Improvements in product quality tend to be the goals of process improvement. This is not surprising since computing elements and their associated software are becoming ubiquitous in the things we used daily. Properly working software is thus an extremely important goal

- most pi is guality improvement

- growing segment where quality is adequate for market, but costs are too - high and intervals too long – hence reduce costs/intervals

- one way of doing this is to remove sequencing and synchronization points

- two large contributors

- - - and process overhead - iso requirements

- - - temporal and geo dislocation,

Process Overhead

- paper trail - iso reqs

- distribution

Temporal and Geographical Dislocation

An increasingly popular trend in large-scale software development is the use of development teams that are geographically separated. Instances of this trend range from groups that are contained in multiple buildings to groups that are located in multiple continents. While the former tend to separated only geographically, the latter tend to be separated temporally as well. While geographical separation tends to encourage asynchronous activities because of cost factors, temporal separation often precludes synchronous activities because of non-overlapping work hours.

It is in this context that the dissemination of critical information and the synchronization of coordinated activities are critical problems. While these problems are not insurmountable, their solutions have varying trade-offs in terms of time, cost and effectiveness. These solutions range from the simple form of using speaker-phones to the complex form of multimedia with technologically intensive computer-supported cooperative work — that is, from relatively inexpensive and primitive solutions to expensive and sophisticated (but as yet experimental) solutions. Note, however, that temporal separation tends to make these synchronized solutions usable only for short periods during the workday at best and completely impracticable at worst.

These two forms of separation can introduce or exaccerbate bottlenecks in project schedules. Our previous studies [?] have shown that the inspection interval is typically lengthened because of schedule conflicts among inspectors which delay the (usually) required inspection collection meeting. This problem is intensified in geographically and temporally distributed settings.

EXPLORING ALTERNATIVES

-small intro

The Solution Space

- requirements to satisfy: reduce cost/interval

- responses

- - - reduction of paper -

- - - generating iso records and measurement info

- - - remove synchronization - reduce coordination

- what are the design alternatives

- - - obvious things - put it on the web, gen records, etc - just go with it

- - - need to think hard about - synchronization - analysis

- - - - - removing synch and compartmentalization - how to do that

- - - - meetings or not - synch vs asynch

- - - - - get rid of synchronization and make concurent

- - - - - larrys analysis

- - - - shared vs private preparation

- - - - - shared easier given web ability to disseminate info

- - - - - shared captures some aspects of meetings - knowledge of what others think

- - - - - dont understand effect - is there bias, inhibiting, enhancing??

- - - - - argue it is a second order effect at worst, helpful at best

- - - - overlap of review and repair

- - - - - data analysis suggests that overlap in desk check

already occurs

- - - - - possible as a further improvement, but dont understand well

- - - - - currently not done (much) - sequentialization

the former is something we did, but still do not uderstand well, the latter is something that is possible, but which we have not explored the consequences of yet. it is a result of the design choice for the former

Evaluation and Justifica tion
- results of data analysis

- - - source of data: controlled experiments, no experiment

- - - desk vs meetings - hyp desk no worse than meetings

- - - live experiment - developers in control - data

- - - - - basic information, details

- - - - - average fault density is virtually identical

- - - - - desk inpsections slightly better - stat sig difference, but small

- - - - - dont know selection criteria - but as good as random since results are identical - how decision was made - ie tested potential biases and didnt find them - no more effective than random

- - - - - whatever caveats we need

- old section follows

Although we have been unable to conduct a controlled experiment to compare the effectiveness of these two inspection approaches. We do have other data that sheds some light on this topic.

If on-line inspections are better than manual inspections, then it must be possible to eliminate meetings without decreasing effectiveness. Previous work [?, ?, ?] suggests that this is indeed the case, but until now there has been no direct evidence from an industrial environment.

To answer this question we are exploiting a natural experiment currently running at Lucent Technologies. (see [?] for a similar example). The advantage of this is that the empirical infrastructure is already in place; that is the software development organization was already measuring the effects of two different inspection processes (desk-based collection versus meeting-based collection) and recording critical data for the two processes. Hence, there was no intrusion on the part of the experimenters and our role was that of interpretation.

We compare the results of two classes of inspections: new code (Table 1) and repaired code (Table 2). The significance is calculated using the Wilcoxon-Mann and Whitney Rank Order Test [?] , a two-sided test assessing whether the fault densities observed for each inspection when taken from a desk or meeting are drawn from the same distribution. The smaller the number, the more significant: numbers of .1 to .05 indicate a mild significance and numbers below .05 indicate significance.

To determine whether the asynchronous desk inspections are as effective as the meeting collections, we look at inspection statistics taken from almost 3000 inspections conducted in this environment. Table 1 and Table 2 show these statistics for new and modified code respectively.

The Tables show that there is now difference is the average fault density of new code inspections found by desk inspections or meeting-based inspections. There is a significant difference for modified code, but the difference is effectively 0. (.0031 vs. .0037). Since this is and order of magnitude smaller than the densities for new code we conclude that meetingless inspections are no less effective than inspection with meetings.

Moreover, there is very little difference in the time needed to repair new code, though the slightly less time take might be due to overlapping repair with collection.

Summary of Design Choices
Given the geographical and temporal separation of many of our projects, it is immediately obvious that electronic distribution saves both delivery time and distribution costs, especially when several continents are involved.

dont expect it to change the way select reviewers, org meetings etc

What has not been taken advantage of is the possibility of further concurrency in the inspection process — namely, that the resolution and repair phase can proceed concurrently with the inspector preparation and collection phase (probably because work patterns are hard to change). While there are undoubtedly cases where defects interact and the expense of coordinated changes is less than separate changes, in most cases the changes are independent and hence concurrent repair would be cost effective[1].

While it is clear that the new approach is less expensive, we do not know if it is also less effective.

We present and justify a solution using an intranet web that is both timely in its dissemination of information and effective in its coordination of distributed inspectors. First, exploiting a naturally occurring experiment (reported here), we conclude that the asynchronous col-

[1] In software developments where the fault density is higher before inspections, this may not be a good assumption.

	Desk	Meeting	Both	Significance
Number of Inspections	202	441	643	NA
Average Faults/Inspection (Faults)	10.1	8.8	9.2	.20
Average Code Size/Inspection (NCSL)	427	327	358	.02
Average Fault Density/Inspection (Faults/NCSL)	.030	.029	.030	.92
Average Repair (Days)	7.1	8.0	7.7	.10

Table 1: Comparison of Desk and Meeting Inspection Detection Effectiveness for New Code.

	Desk	Meeting	Both	Significance
Number of Inspections	2152	197	2152	
Average Faults/Inspection (Faults)	.163	.432	.185	.002
Average Code Size/Inspection (NCSL)	26.0	59.4	28.8	—
Average Fault Density/Inspection (Faults/NCSL)	.0031	.0037	.0031	.03
Average Repair (Days)	1.2	3.3	1.3	NA

Table 2: Comparison of Desk and Meeting Inspection Detection Effectiveness for Repaired Code.

lection of inspection results is at least as effective as the synchronous collection of those results. Second, exploiting the information dissemination qualities and the on-demand nature of information retrieval of the web, and the platform independence of browsers, we built an inexpensive tool that integrates seamlessly into the current development process. By seamless we mean an identical paper flow that results in an almost identical inspection process that is consistent with ISO certification.

BUILDING AND EVALUATING IMPROVEMENTS

short forecast/intro

THE `hyperCode` SYSTEM

fix next paragraph

We discuss two basic views of `hyperCode`: the process view and the implementation view. In the first, we discuss the observable characteristics of the tool and how they affect the authors, moderators and inspectors. In the second, we discuss various details of how we make things happen, either directly or indirectly.

The `hyperCode` Inspection Process

1. Modification Requests (MR's) are issued whenever additions or enhancements to code are needed.

2. A developer accepts one or more MR's and develops the necessary code.

3. The author then makes a code unit available for inspection by interacting with the `hyperCode` tool.

4. The author selects his or her review team, again by selecting their names from a `hyperCode` form.

5. `hyperCode` then contacts the review team and project management who respond to schedule the closing date of the inspection. (There is no meeting in the `hyperCode` process).

6. `hyperCode` prepares the inspection package and notifies the review team of the package's location.

7. Prior to the meeting, the reviewers analyze the code unit looking for defects. Reviewers analyze the code concurrently, while `hyperCode` automatically collecting all annontations.

8. Once the inspection is closed the author receives the consolidated list of issues from the `hyperCode` system.

9. The author determines which issues must be repaired, and does so.

10. The author brings the reworked code to the inspection moderator who ensures that all issues have been addressed and signs off the inspection.

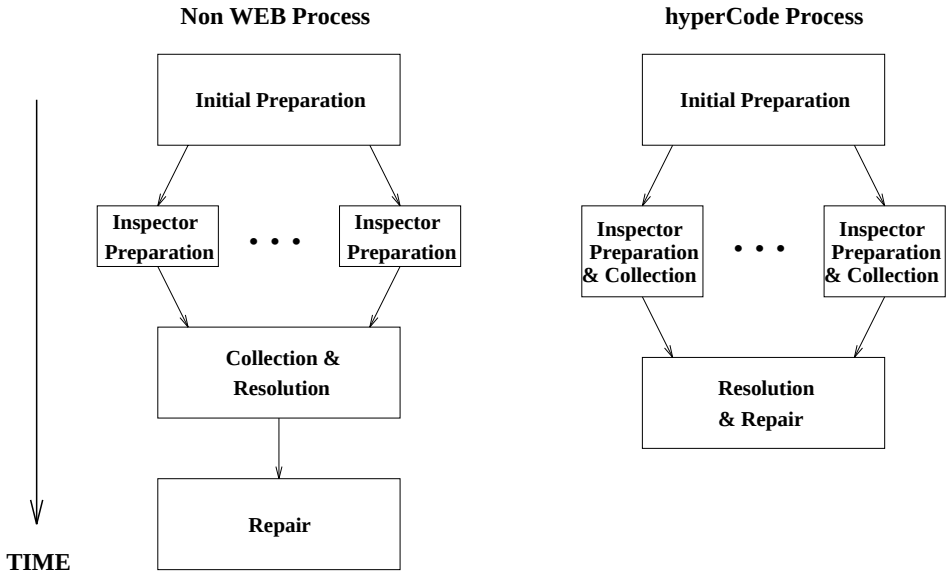

Figure 1: Comparison of the Inspection Processes.

The primary differences between the manual and `hyperCode` processes are as follows.

- Automated support for inspector selection.
- Automatic notification by e-mail that the package is available.
- All annotations are visible to all reviewers and the author throughout the process (concurrent preparation) .
- There is no meeting in the `hyperCode` process (asynchronus team interaction).

User Interface

`hyperCode` is a web-based code inspection system. During a designated inspection interval, inspectors use the Netscape Navigator web browser at their desktop computers to view and annotate the code under inspection (see Figure 2 for an example of the user interface). All annotations are viewable by all participants. This inspection process does not require the simultaneous participation of the inspectors, nor do inspectors need to be geographically co-located. All that is required for participation is access to the intranet via the Netscape Navigator web browser. At the end of the inspection interval, the author and moderator resolve inspector annotations and the author makes code changes as appropriate. All aspects of the code inspection are performed via web pages. E-mail notification replaces paper meeting notices, status reports, etc.

`hyperCode` makes use of an already existing tool that generates code inspection packages (see Figure 3). The essential part of the code inspection package is a diff-marked code listing that highlights new and modified lines of source code. Traditionally, this code inspection package is printed on paper and distributed to the inspectors. A `hyperCode` web-based inspection package is generated by running the output of the already existing inspection package generation tool through a filter that generates an HTML version of the package (line numbers become hyperlinks that provide the ability to annotate, page numbers in the table of contents become hyperlinks to the corresponding pages, etc.).

The `hyperCode` inspection package has the same layout as the paper version - experienced developers are therefore immediately familiar with `hyperCode` inspection packages. The ability to create and view inspection packages, create and manage annotations, send e-mail notifications, etc. are provided by a set of CGI scripts maintained at the webserver. No special purpose software is needed by users of `hyperCode` - the only software required of users is the Netscape Navigator web browser (since `hyperCode` makes use of frames, Netscape Navigator version 2.0 or later is required).

An author creates a `hyperCode` inspection package by bringing up the package creation web form and entering information about the package, including the usernames of those who are to be inspectors. The author also designates one of the inspectors to be the moderator of the inspection. Standard WWW username/password authentication is used to identify users and control access. The author then submits the form, which causes the webserver to invoke the standard inspection generation tool and feed the results to the HTML filter, the output of which is the `hyperCode` inspection package which is

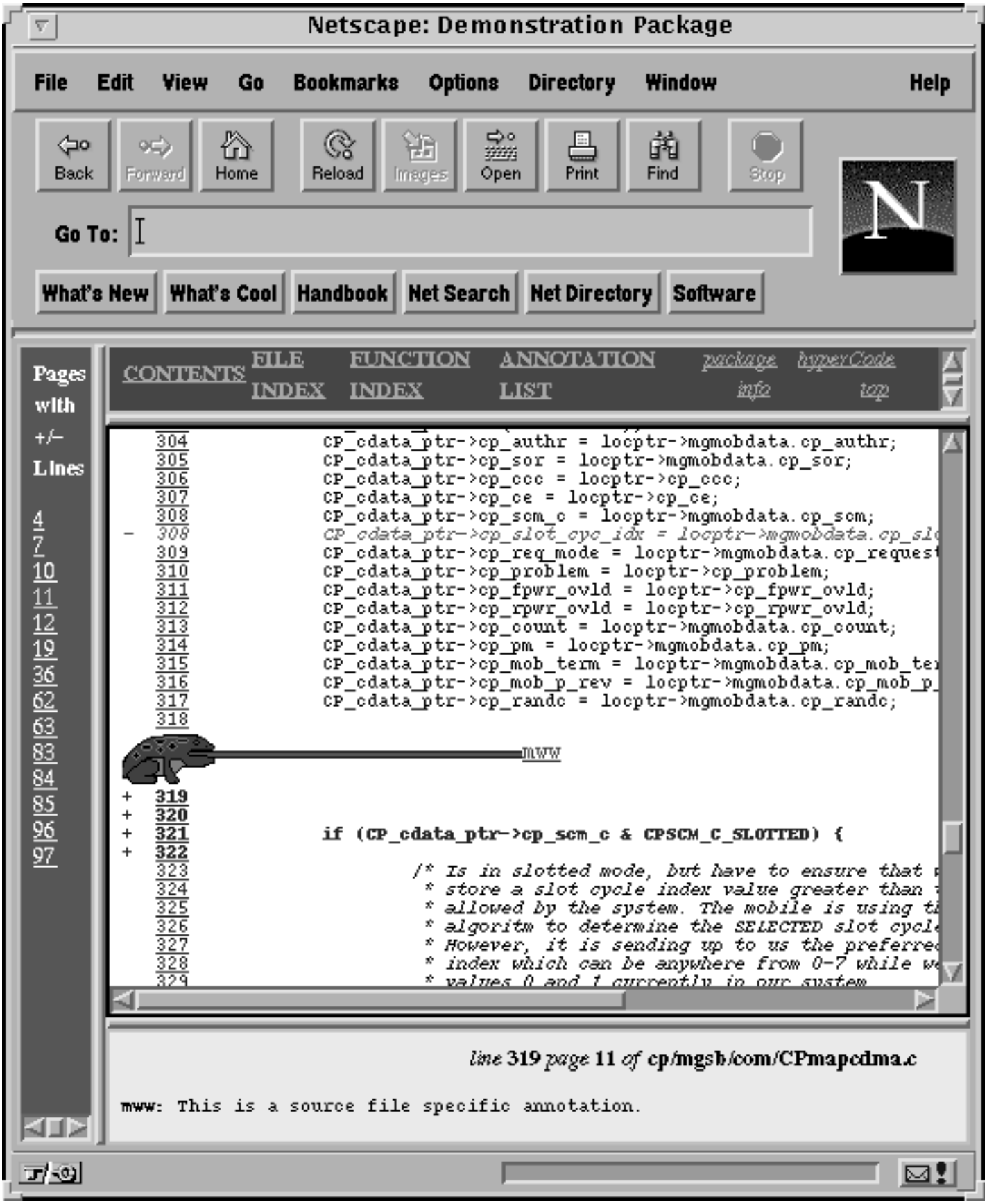

Figure 2: Example of the User's View of hyperCode.

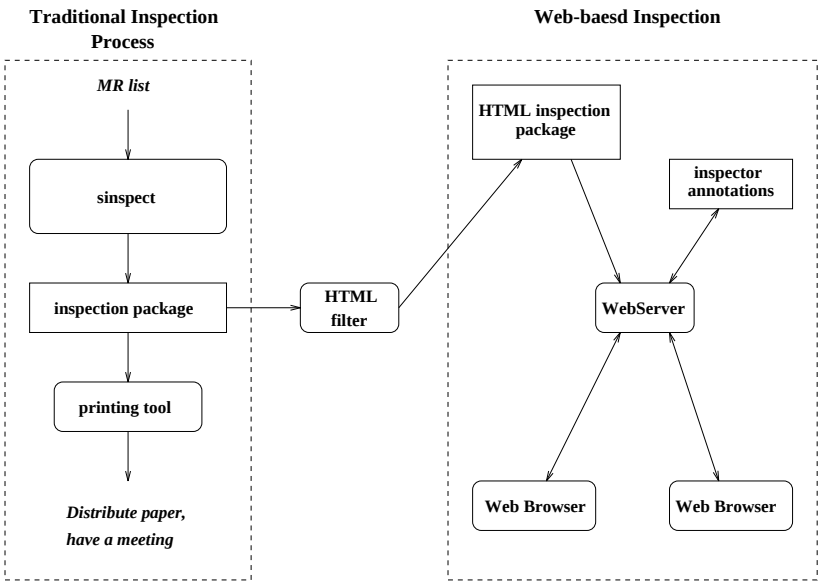

Figure 3: Generating the Inspection Packages.

deposited in a node managed by the webserver.

A `hyperCode` inspection package goes through a lifetime consisting of 4 states: pending, in progress, resolution, and done. Packages can be viewed in any state, but annotations can only be made by the inspectors when the package is in the in progress state. A package is initially created by the author in the pending state. The author then moves the package to the in progress state, which causes e-mail notification to be sent to the inspectors and other interested par- ties (project management, quality team, etc.). The designated inspectors may now inspect the code and make annotations.

At the end of the designated inspection interval, the author moves the package to the resolution state. This state transition again generates e-mail notification to the inspectors and other interested parties. The author then determines the disposition of each annotation and records (via `hyperCode` web page) whether any code changes will be required. After the disposition of all annotations has been determined, the author then informs the moderator via e-mail that the package is ready for moderator sign-off. The moderator then verifies the disposition of the annotations.

The moderator then moves the package to the done state. This state transition generates a final e-mail notification to inspectors and other interested parties.

Implementation

Source code line numbers are hyperlinked to a form that allows inspectors to enter annotations. That is, when an inspector clicks on a source code line number, a web form containing a text input area is presented. The inspector enters the annotation and submits the form, which causes the webserver to make a record of the annotation. The record contains the username of the inspector, the line number and source code file name, along with the text of the annotation.

For each inspection package, `hyperCode` provides a page that lists all annotations that have been made to date by the package inspectors. The annotation list contains hyperlinks to the annotation text and to the relevant source code page. The annotation list is ordered by source file and line number. The annotation list page is generated via a CGI script, so the page is up to date each time it is reloaded by a web browser.

If a source code line has been annotated by an inspector, a graphical element appears in the left hand margin of the source code display page as a visual cue to inspectors or other viewers of the package. The graphical element is hyperlinked to the text of the corresponding annotations.

In addition to source file-specific annotations, inspectors may also make general annotations that do not refer to any particular line of source code in the package. These type of annotations may be used to record general concerns or issues that are global to the source code under inspection. At the top of each source code display page is a hyperlink to a web form that allows these types of annotations to be made. General annotations also appear on the annotation list page.

Evaluation of Initial Experience

The acceptance of the inspection tool has been excellent. We attribute this to four basic facts: First, the cost savings just from the reduction in paper work and the time savings from the reduction in distribution interval of the

inspection package (sometimes involving international mailings) have been substantial. Second, the new intranet tool-based process integrates seamlessly into the existing environment and workflow. This point is both a subtle and a critical one. The disruption of existing workflow almost always causes both resistance and unexpected side-effects. Third, the new process opens up new possibilities for concurrency and inherent speedups of the elapse time interval. Fourth, the ubiquity of the web with its distribution and random accessibility as well as its browser platform independence makes it a natural platform for such an approach as ours.

SUMMARY

Conclusions

- cost effective

- shorter interval

- - - remove compartmentalization, sequencing

- - - at least as effective

While there has been much work on inspections structures, inspection techniques and automated inspection support, we believe we are the first to report on the use of an intranet-based tool to support asynchronous (that is, meetingless) code inspections. The primary effort in prior automation is in the application of CSCW support for inspection collection meetings — that is, in the support for synchronous meetings (see for example [?, ?]). But as we have shown above, asynchronous code inspections are more cost effective and at least as quality effective as synchronous inspections. Moreover, the cost of asynchronous automated support is significantly less than that of synchronous.

The empirical data we report here is the first such data showing specifically that asynchronous code defect collection is as effective as the synchronous code defect collection.

Open Questions

- effects of shared vs private –

- - - argue from siy thesis, no worse

- - - sharing incorporates (good?) aspects of current meetings

- - - prelimary data looks encouraging

- overlap of review and repair

- - - consistency

- - - premature repair, repair on repair (potential for code decay)

- - - possibly more effort, certainly shorter interval

- - - serious study needed here for effects

- analysis to show that overlap looks like a good idea