Understanding the Effects of Developer Activities on Inspection Interval

Adam A. Porter[*], **Harvey P. Siy**
Computer Science Department
University of Maryland
College Park, MD 20742 USA
+1 630 224 6830
{aporter,harvey}@cs.umd.edu

Lawrence G. Votta, Jr.
Software Production Research Department
Lucent Technologies
Naperville, IL 60566 USA
+1 630 713 4612
votta@research.bell-labs.com

March 6, 1997

1 ABSTRACT

We have conducted an industrial experiment to assess the cost-benefit tradeoffs of several software inspection processes. Our results to date explain the variation in observed effectiveness very well, but are unable to satisfactorily explain variation in inspection interval.

In this article we examine the effect of a new factor – process environment – on inspection interval (calendar time needed to complete the inspection). Our analysis suggests that process environment does indeed influence inspection interval. In particular, we found that non-uniform work priorities, time-varying workloads, and deadlines have significant effects.

Moreover, these experiences suggest that regression models are inherently inadequate for interval modeling, and that queueing models may be more effective.

1.1 Keywords

Software inspection, empirical studies, statistical modeling, interval reduction, queueing.

2 INTRODUCTION

Companies that cannot build quality products as quickly as their competitors may find themselves at a severe competitive disadvantage. Therefore understanding, identifying, and eliminating bottlenecks in software development is extremely important. Until recently, however, little research has addressed this issue.

Previously we conducted an industrial experiment at Lucent Technologies to determine which factors drive the cost and benefits of different software inspection processes [8]. To date we have explored the following factors.

- process structure (e.g., team size, number and sequencing of sessions),
- process techniques (e.g., preparation times, inspection rates), and
- process inputs (e.g., reviewers, authors, code quality).

Using regression analysis, we found that although these factors explain much of the variation in effectiveness, they do not adequately explain variation in inspection interval [7].

In this article we examine whether a fourth factor – process environment – explains this variation. The process environment is the logistic, organization, and execution context in which a process operates.

We conjecture that process environment affects interval when development processes subtly influence one another. These influences manifest themselves in several ways.

[*] This work is supported in part by a National Science Foundation Faculty Early Career Development Award, CCR-9501354. Dr. Siy was also partly supported by AT&T 's Summer Employment Program

- The coding process provides input to the inspection process; code must be written before it is inspected. However, when many inspections are in progress, there's little time left to write new code. So inspections affect coding as well.
- Inspection tasks are always interleaved with other development processes – writing other code, developing tests, inspecting other units, etc. Therefore, the completion of an inspection may depend on the completion of a seemingly unrelated task.
- Developers often work on multiple projects simultaneously. This means that working on one project's task may delay the completion of another's.
- Coding assignments are not always distributed uniformly in time or uniformly throughout the development team. One developer may build an entire subsystem early in the project while another builds a different subsystem later. Consequently, coding assignments may come in bursts.
- Most tasks have some deadlines. As they approach, some tasks whose deadline is not near may be deferred to a later date.

Through direct observation and surveys we found that developers often have to choose which of their many activities to perform at any given time. We hypothesize that the process environment influences these choices and that they, in turn, influence inspection interval.

Our analysis suggests that process environment does indeed influence inspection interval. In particular, we found that different coding and inspection tasks have different priorities. Therefore, when a developer's workload is high, low priority tasks are deferred. Since some inspection tasks have very low priority, this lengthens inspection interval.

Moreover, this situation suggests that regression modeling is inadequate for intervals. Instead we must investigate models that capture time- and workload-dependent behaviors. Priority queueing networks are a good example of such models.

Below we review our previous research, describe process environment and develop several measures of it, analyze our data to find support for our hypotheses, present a queueing model formulation of this inspection interval, and discuss future research.

3 BACKGROUND

The software inspection process has three steps, Preparation, Collection, and Repair. First, each reviewer individually analyzes the document looking for its defects (Preparation). Next, these defects are collected and discussed, usually at a team meeting (Collection). Finally, a list of known defects are presented to the document's author, who fixes them (Repair).

With the cooperation of professional developers working on an industrial software project at Lucent Technologies, we conducted a controlled experiment to compare the costs and benefits of several different software inspection processes (see Porter, et al. [8] for details). The project was to develop a compiler and environment to support developers of the 5ESS$^{\text{TM}}$ telephone switching system. The complete system contains over 55K new lines of C++ code, plus another 10K which was reused from a prototype.

Our inspection pool consisted of 11 experienced developers[1], each of whom had received inspection training within the previous five years. The experiment ran for 18 months during which 88 code inspections were performed.

We manipulated several independent variables including, the number of reviewers (1, 2, or 4), the number of sessions (1 or 2), and, for multiple sessions, whether to require or prohibit repair of known defects prior to holding the second session. A treatment is specified by assigning a value to each of these three variables. For example, one treatment involves 2-sessions, with 2-**p**ersons per session with **R**epair in between the first and second sessions. This is denoted 2sX2pR.

For each inspection our dependent variables included observed defect density (effectiveness), and working days to complete (interval). Whenever a new code unit became available for inspection, it was randomly assigned a treatment and a set of reviewers. In this way we attempted to control for differences in natural ability, learning rate, and code quality.

[1]In addition, 6 more developers were called in at one time or another to help inspect 1 or 2 code units, mostly to relieve the regular pool during peak development periods. We did not include them here because they contribute too few data points.

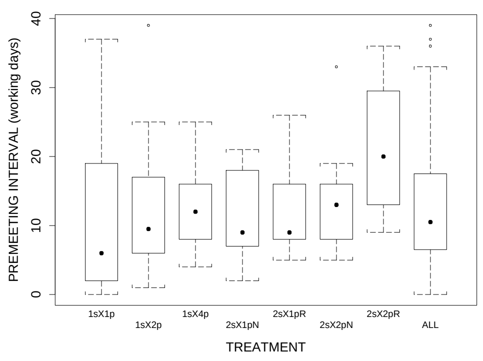

Figure 1: **Pre-meeting Interval by Treatment.** The distributions are similar except for 2sX2pR, which was significantly higher.

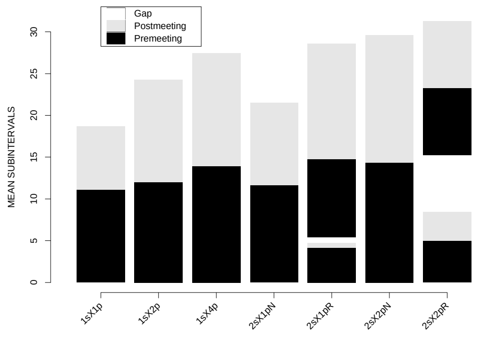

Figure 2: **Subinterval by Treatment.** This stacked barchart depicts the mean pre- and post-meeting interval by treatment. Note the pronounced gap between the first and second sessions of 2sX2pR inspections.

Our initial results showed the process structure (number of reviewers, the way they are organized, etc.) doesn't influence effectiveness. However, repairing defects in between multiple sessions did significantly increase interval.[2].

Figure 1 depicts the distribution of pre-meeting inspection intervals by treatment. The pre-meeting interval is time in working days from the time the code unit was ready for inspection to the time of the Collection meeting. Our initial analysis focused on pre-meeting rather than total interval because authors sometimes deferred repair, which inflated the total interval.[3]

We saw that most of the distributions are similar except for the 2sX2pR treatment. Although, our statistical analysis indicated that repairing defects in between two sessions significantly increased interval, Figure 2 shows that the difference is really due to a time lapse between the end of the first session and the start of the second. Furthermore, this gap does not appear in 2sX1pR inspections, which suggests that repair does not necessarily increase interval. One interpretation is that multiple sessions increase interval as the number of reviewers increases (possibly due to scheduling difficulties).

When we consider total interval, there are no significant differences due to the treatments, but there is still a huge amount of variation within them. Therefore in the remainder of this article we will consider total interval

[2]In this experiment, we consider two data distributions to be significantly different only if the Student's t and the Wilcoxon rank sum test both reject the null hypothesis that the observations are drawn from the same population with a confidence level ≥ 0.9, i.e., $p_t < 0.1, p_w < 0.1$. In most cases, the two tests agree and when they don't agree, it is usually the case that one is near the borderline.

[3]Each session of a two-session inspection has its own interval. We calculate the entire inspection's pre-meeting interval as follows. For inspections without repair, it is the longer of the two pre-meeting intervals, since both begin at the same time. For those with repair, it is the two sessions placed end-to-end, excluding the post-meeting interval from the second session.

and attempt to explain its variation.

A final issue is that we have chosen to analyze each session of a 2-session inspection separately. This simplification allows us to model one- and two-session inspections uniformly.

4 MODELING INSPECTION INTERVAL

In order to understand the effect of process environment on inspection interval, it is important to have a detailed understanding of the inspection process and its interface to the coding process. In our environment developers enacted the following process.

4.1 The 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. A code unit may implement one or more MR's.
4. A particular inspection treatment and a review team is randomly assigned to the inspection.
5. The author contacts the review team and schedules the inspection meeting. (If the treatment calls for 2 sessions with no repair in between, the author contacts 2 sets of reviewers and sets up two separate meetings.)
6. Prior to the meeting, the reviewers analyze the code unit looking for defects.
7. 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.
8. After the meeting the author collects the consolidated list of issues. Issues are the potential defects discovered during the inspection.
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.
11. If the treatment calls for 2 sessions with repair in between, then the author repeats all inspection steps one more time.

4.2 Analysis Strategy

The goal of this analysis is to determine whether the data supports our hypothesis that the interaction of process environment and task priorities explain variation in inspection interval.

We analyzed inspection interval using linear regression models [2]. We built one model for the pre-meeting interval (time from availability to meeting) and another for the post-meeting interval (time from meeting to the completion of repair). Our reasoning was that since only one reviewer (the moderator) is involved after the meeting, it is likely that different factors come into play during each of these two inspection phases.

4.3 Potential Sources of Variation

The factors we investigated captured information about the process structure, the process inputs, the process techniques, and the process environment. Data for these factors were extracted from the inspection data collected for the experiment and the change management database being used by the developers.

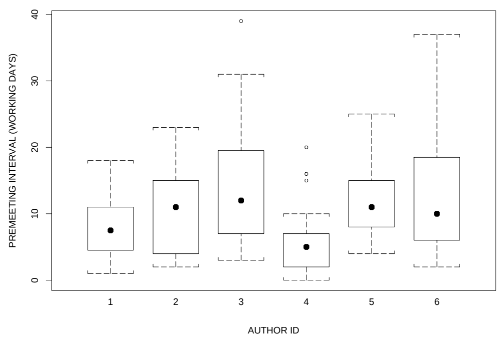

Figure 3: **Effect of Authors on Pre-meeting Interval.** This boxplot show the effect of authors on pre-meeting interval.

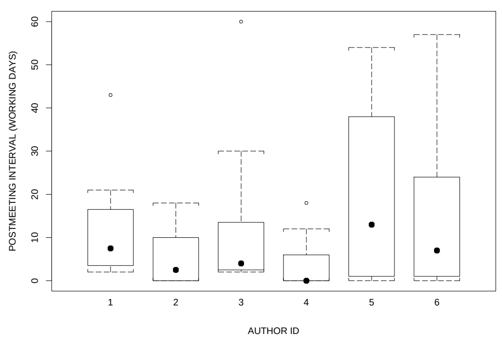

Figure 4: **Effect of Authors on Post-meeting Interval.** This boxplot shows the effect of authors on post-meeting interval.

4.3.1 Process Structure

Process structure factors describe the manner in which inspection steps and resources are organized into a process. In this study we examine three such factors: Team size, number of sessions, and repair policy. (These variables are described in the BACKGROUND Section.) Team size and number of sessions have no significant effect on interval; As described earlier, the sessions of a two-session with repair inspections tend to have a smaller than average interval.

Because we plan to model each session of two-session inspections separately, we must take care not to ignore possible effects of interactions in between two-session inspections. Hence, we will add variables to tell us whether a particular session is the first or second session and whether there was repair in between or not.

4.3.2 Process Input

Process inputs describe the raw materials used to conduct the inspection. This includes the code itself and the various participants.

Code Size. The size of a code unit is given in terms of non-commentary source lines. Code size does not have a significant effect on interval.

Author. The author is the central person in the inspection. He or she writes the code being inspected, coordinates the inspection, and resolves and repairs the issues that are raised. As shown in Figures 3 and 4, the effect of the unit's author is significant.[4]

Reviewers. The reviewers in our study are labeled A through K. Reviewers A through F are members of the development team and are called internal reviewers. Reviewers G to K are not team members and are called

[4] Significance tests for multiple comparisons like this were computed using the F-test.

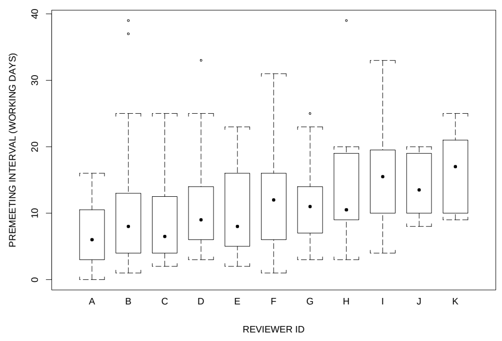

Figure 5: **Effect of Reviewers on Pre-meeting Interval.** This boxplot shows the effect of reviewers on the pre-meeting interval. Reviewers do not have a significant effect on post-meeting interval.

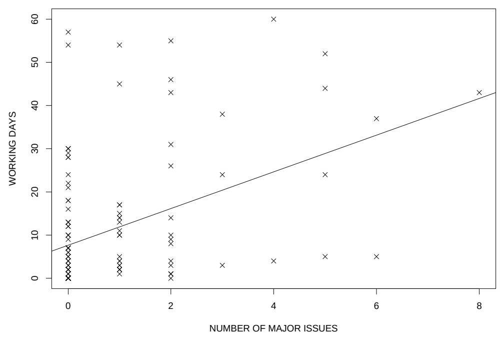

Figure 6: **Effect of Major Issues on Post-meeting Interval.** This plot shows the effect of major issues on the post-meeting interval.

external reviewers. Reviewers have a significant effect on the pre-meeting interval, as shown in Figure 5. They have no effect on the post-meeting interval.

4.3.3 Process Techniques

Process techniques refer to the technical activities, the methods and outcomes of inspection itself. We examined the following aspects of process techniques.[5]

Major Issues. This is the number of issues recorded during the inspection, whose repair required more than four hours to complete. Figure 6 shows that major issues significantly affected the post-meeting interval.

Total Issues. The total number of issues recorded during the inspection. One might expect that a greater number of issues would take longer to resolve, however, the effect on post-meeting interval is not significant.

True Defects. The total number of issues that the author repaired and whose repair affected the execution behavior of the system. This is contrasted with issues that involved coding standards, documentation, etc. The number of true defects did not have a significant effect on post-meeting interval.

4.3.4 Process Environment

The process environment factors we examined include measures of workload, lifecycle phase during which the code is inspected, and the presence of deadlines.

Workload Measures. The workload measures are an estimate of how busy a developer might have been at the time of the inspection. We calculated several workload measures by summing the number of pending inspections

[5] Since issues are raised before or during the inspection meeting, these measures only apply to the post-meeting interval.

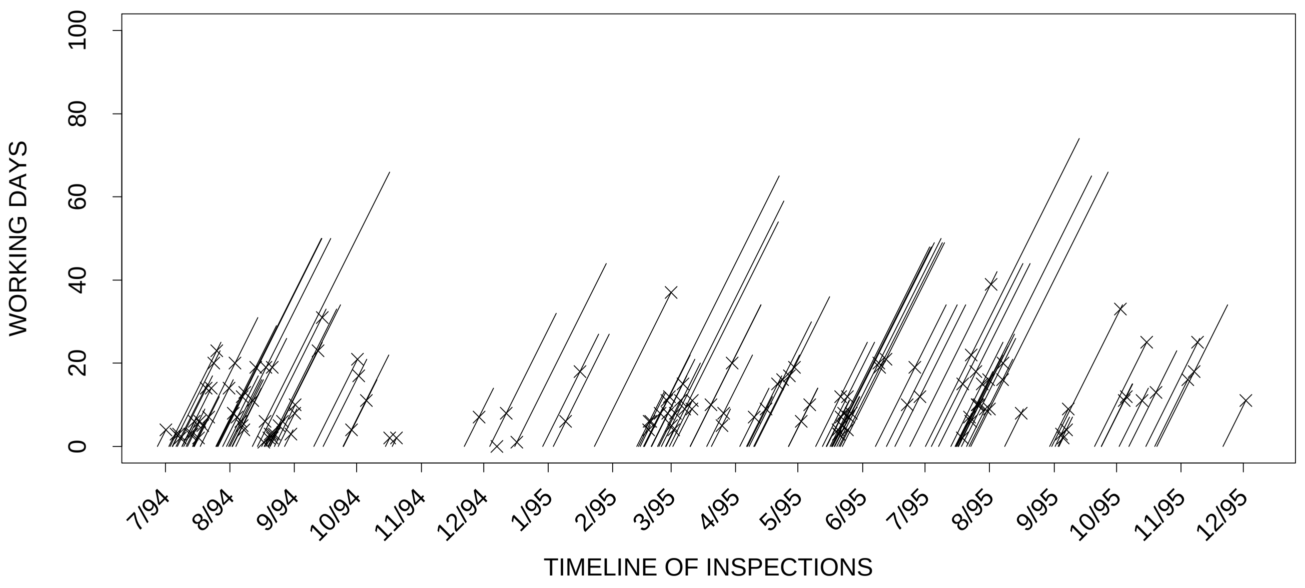

Figure 7: **Timeline of Inspection Activities.** This plot is a timeline representing the inspection tasks that occurred during the study. Each diagonal line represents one session. The lower end of the line indicates the start of the inspection. The line's length is proportional to the inspection's interval. Each line contains an "X" which marks the point in time when the inspection meeting occurred. Note that inspections often come in bursts and that during these periods repair is frequently deferred.

(*inspection load*), inspections with unfinished rework (*rework load*), and pending MR's (*coding load*). Figure 7 is a timeline of inspections over the duration of the study. This figure points out that rework tends to be deferred when many inspections are taking place.

We calculated pre-meeting workloads for the 2-week period spanning the week before and after the code unit became available. For post-meeting workloads, we considered the week before and after the inspection meeting.

We calculated workload measures for the author and for the inspection team. Since the busiest person is often the bottleneck in scheduling, the workload measures for the inspection team were the maximum scores of each task type among the participating reviewers. Reviewer workloads did not have a significant effect on pre- or post-meeting interval. The plots in Figure 8 show the effect of author workload on pre-meeting and post-meeting intervals.

Month Into Project. Through the 18 months when data was collected, the overall mean time to complete an inspection may change, i.e., may have a tendency to increase or decrease over time. Figure 7 does not show any increasing or decreasing trend in the length of the inspection intervals.

5 REGRESSION ANALYSIS

We built models of the pre- and post-meeting intervals using factors from the process structure, process inputs, process techniques, and process environments which significantly explain their variance. More details on model building can be found in the appendix.[6]

5.1 Pre-meeting Interval

The following model of pre-meeting interval explains 35% of the variance using only 10 out of 130 degrees of freedom.

$$\begin{aligned} Premeet &\sim Author + ExternalMember + \\ &\quad AuthorCodingLoad + \end{aligned}$$

[6]For example, the section entitled "Calculating the Significance" describes the test determining which factors are considered significant to the model.

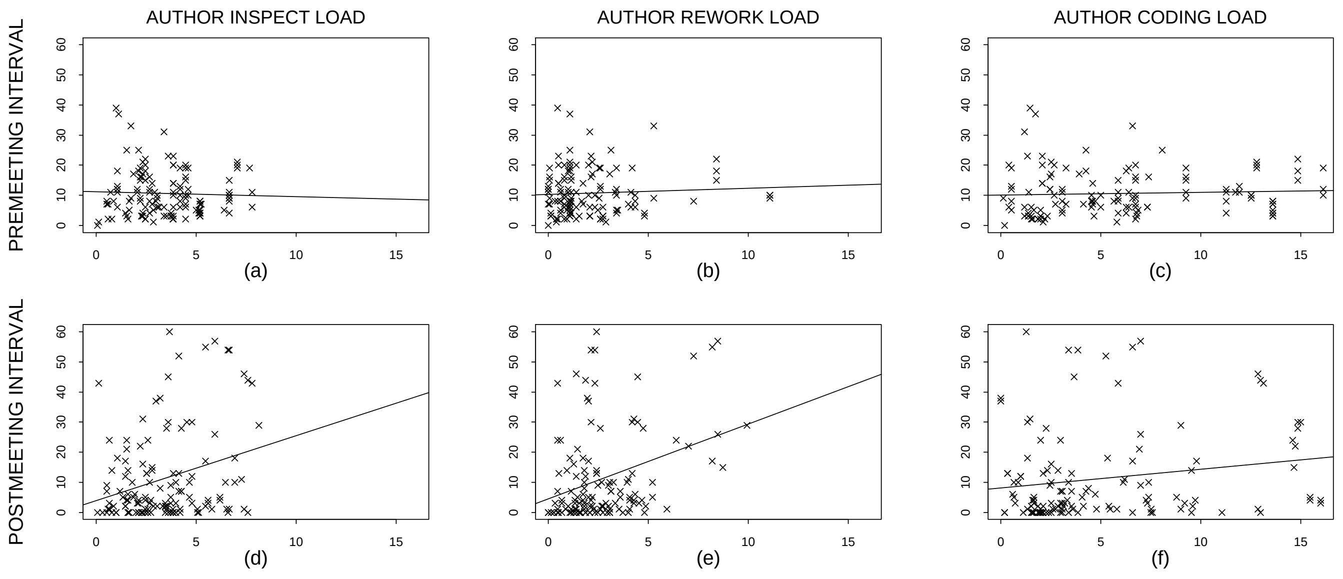

Figure 8: **Effect of Author's Workload on Interval.** This matrix of plots shows the correlation between some inspection interval and some measure of workload. The plot has two rows and three columns. Plots in the upper row are for pre-meeting intervals, plots in the lower row are for post-meeting intervals. Reading left to right, the columns represent the author's inspection load, rework load, and coding load.

$$
\begin{gathered}
1stSessionRepair + \\
2ndSessionRepair + \\
1stSessionNoRepair
\end{gathered}
$$

As we mentioned previously, repairing defects in between two-session inspections affects interval, but this is because the pre-meeting intervals of the sessions involving repair are significantly less than that of the rest of the treatments (see Figure 2). In addition, the first session of the inspections with no repair is also significantly less than the rest of the treatments. These indicate the presence of an implicit deadline to get all sessions of an inspection completed in a reasonable amount of time.[7]

The code unit's author also affects the pre-meeting interval, as shown in Figure 3. This possibly occurs because he or she initiates the inspection process by making the code unit available for inspection and coordinates the scheduling of the inspection meeting.

The presence of at least one external reviewer significantly increases the pre-meeting interval. This may be because most of the external reviewers are not immediately available for inspections and take longer to schedule. Figure 5 shows that, with the possible exception of Reviewer G, inspections involving external reviewers have higher interval distributions than those involving internal reviewers.

The model shows that reviewer and author inspection workloads do not explain the variance in pre-meeting interval. However, the author's coding load – number of pending MR's – is significant, but it turns out to be a negative contributor to the interval! Figure 8(c) shows the negative relationship between author coding load and pre-meeting interval.

5.1.1 Residual Analysis

A model is adequate if it reasonably estimates the data and its residuals are just "white noise," i.e., there is no detectable pattern in the residuals. Figure 9 gives a graphical test of these two conditions. The plot on the left compares estimated with the original values. The correlation between them suggests that the model reasonably estimates the original data. The plot on the right compares estimated values with the residuals. The variance in

[7]In the case of two-session inspections with no repair, we took the convention of labeling the session whose meeting occurred earlier as the first session. This implies that (1) the first interval will always be less than the second interval, and (2) if there is an implicit deadline, the second session's meeting should occur at about the same time as for any one-session inspection.

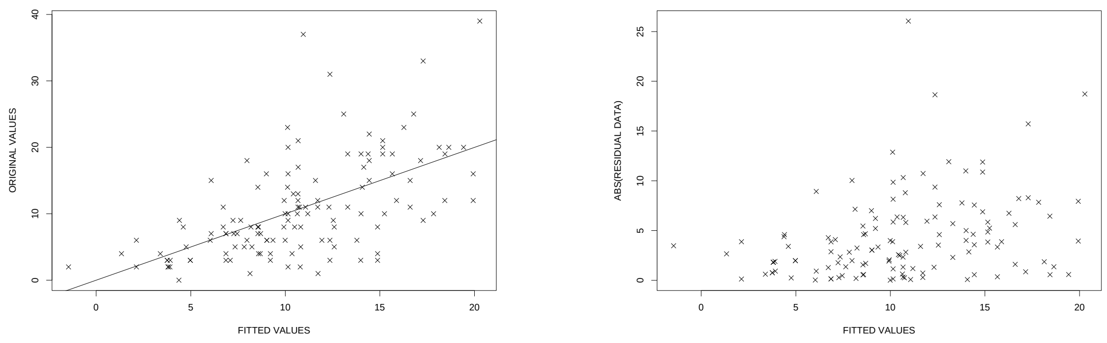

Figure 9: **Examining the fit of the pre-meeting model.** The plot on the left compares the model's estimated values the original values (if the model were perfect all the points would line on the line y = x). There is a substantial correlation between the two (cor = 0.59). The plot on the right compares the estimated values with the residuals. The variance appears to increase as the estimates do. This may suggest non-linear effects.

the residuals appears to increase with the estimated values. This suggests that the functional form of the model (i.e., linear effects) is inadequate.

5.2 Post-meeting Interval

The following model of post-meeting interval explains 32% of the variance using only 8 out of 130 degrees of freedom.

$$\begin{aligned} Postmeet \sim\ & Author + AuthorInspectload + \\ & AuthorReworkLoad + MajorIssues \end{aligned}$$

The number of major issues found during an inspection significantly lengthens the post-meeting interval. This may occur because major issues require the author to reserve a significant block of time (like half a day) just to fix it, so it has to be explicitly scheduled into the author's (normally busy) work schedule. Figure 6 shows the positive relationship between major issues and post-meeting interval.

The author is significant possibly because he or she plays the central part in the post-meeting interval; deciding whether to perform rework immediately, postpone it, or spread it out over time. Figure 4 shows that different authors apparently have different preferences.

The model also shows that author inspection and repair loads increase the post-meeting interval. Figures 8(d) and 8(e) show the positive relationship between inspection and repair loads, and post-meeting interval.

5.2.1 Residual Analysis

Figure 10 gives a graphical test of the adequacy of the post-meeting model. We see that the model reasonably estimates the original data. Again the variance in the residuals appears to be increasing, suggesting that the model is inadequate.

6 INTERPRETATION

In the previous sections we examined the data from a long-term, controlled, industrial experiment in order to model and understand variation in inspection interval. We had previously been unable to explain this variation using information about the process' structure, inputs, and techniques only. Therefore we we added a fourth factor, process environment to our models.

This additional information explains more variation than did the other three factors combined. However, we are still far from explaining the majority of variation in inspection interval. Nevertheless, this exercise has several very important implications.

In particular, it is instructive to compare the pre- and post-meeting models. We find that pre-meeting interval is not significantly affected by the workload of the author nor that of the inspection team. The author's coding

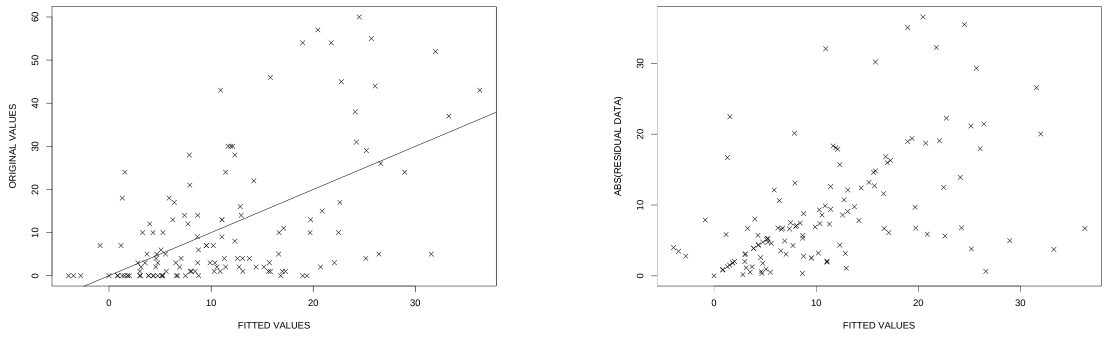

Figure 10: **Examining the fit of the post-meeting model.** There is a correlation of 0.56 between the actual and estimated values. As with the pre-meeting model, the variance of the residuals is positively correlated with estimated values. See the previous Figure caption for a discussion of this plot.

load is significant but it is a negative contributor. This suggests that inspections progress despite increases in the number of code units on which the author is working. These observations imply that authors and reviewers give a higher priority to pre-meeting inspection tasks than they do to pending coding assignments.

On the other hand, the post-meeting interval is significantly affected by pending inspections and rework. This implies that the authors defer rework to complete coding tasks and that rework has a low priority.

Our interpretation of these results is that:

1. Developer workload affects interval. We saw that as an author's coding workload increased, his or her post-meeting intervals grew. Consequently, inspection interval depends on factors outside the inspection process.

2. Developers prioritize their work. At any given time developers may have several unfinished tasks. We saw that even though the author's coding workload increased, their pre-meeting intervals decreased. A conservative interpretation is that pre-meeting inspection tasks are not hindered by pending coding tasks. Another interpretation is that pre-meeting tasks actually delay coding tasks, i.e., they have higher priority.

3. Deadlines alter priorities. We saw that the potentially lengthy two-session inspections had compressed first sessions. This suggests that implicit and explicit deadlines can increase a task's priority.

A final observation is that these factors appear to have complex, non-linear effects and, therefore, linear regression models are probably inadequate for interval analysis. Although problems might lie in poor experimental controls, poor research skills, etc., our findings suggest the need for models that explicitly capture workloads, multiple task types with priorities, and deadlines. Queueing networks are an example of such a model.

7 QUEUEING MODELS FOR INTERVAL ANALYSIS

The idea of modeling software development processes as queueing systems was first proposed by Bradac, et al. [3], but little work has since been conducted on analyzing queues for particular applications. Our application suggests a queueing model in which each developer is a server, handling different tasks with different priorities.

Figure 11 depicts a queueing model of a single developer who performs at least three tasks, reviewing others' code, writing their own, and resolving issues from previous inspections. At any given time, developers can perform at most one task. Based on the previous analysis, these tasks appear to have the given relative priorities.

Figure 12 depicts a global queueing model involving a 3-member development team. New coding tasks enter each developer's queue. When a code unit is finished, it is sent to the inspect queue of the other developers. Once the inspection is completed, the code unit is sent into the rework queue of the original developer. Finally, when rework is done, the code unit exits the system.

This queueing system may be treated as a preemptive priority queue with feedback. Preemptive priority queues are analyzed in Kleinrock [6]. Simon [9] analyzed the case with feedback, where tasks are allowed to feed back into the system and change their priority and service requirements.

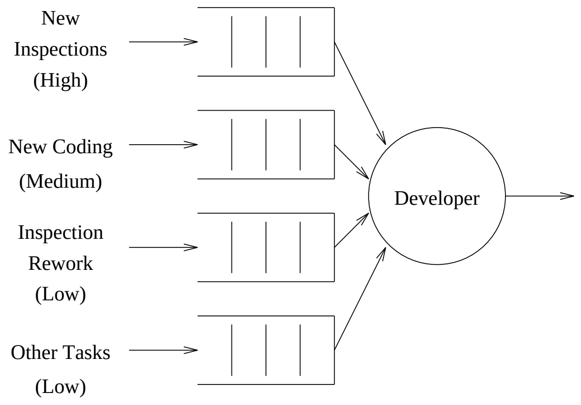

Figure 11: **Modeling a Developer's Task Queues.** This figure illustrates how a single software developer prioritizes and services his or her tasks. New Inspections require the developer to inspect another's code. New Code represents unfinished code the developer is writing. Inspection Rework refers to the uncompleted repair for previously inspected code. The relative priorities of each task type are given in parentheses.

While it is unlikely that a closed form solution can be found, it is still helpful to model the interval process as networks of queues because queues have well-studied properties and relevant dependent variables (time in queue, waiting time, throughput, etc.).

We are currently developing queueing models of inspection interval and attempting to validate them against our data. One interesting statistic that can be derived from these models is the average time a code unit is resident in the system (sojourn time). Our initial analysis indicates that code units enter the system and their development is high priority. After the inspection meeting, however, working on them (reworking defects found in inspection) becomes a low priority. This behavior actually increases the average sojourn time of all code units!

There are several challenges that we must consider in order to apply this approach.

Human Servers. Unlike the uniform servers normally seen in the queueing literature, people have varying abilities and working patterns. They also experience frequent and irregular downtime.

Mapping Code Units to Inspections. Several code units may be collected together and sent as a group to one or more inspection queues. Conversely, one code unit may be split into several smaller pieces, each of which may be inspected independently. The former case is known as bulk arrivals and the latter is known as branching. The presence of bulk arrivals and branching means that tasks may not be independent of each other. There has been some previous work on this problem [9].

Hidden Dependencies. There may be other unknown dependencies between tasks which may cause one to wait on another.

Miscellaneous Tasks. There may be other tasks that consume a significant amount of the developers' time and that must be accounted for.

External Servers. Some inspection work is conducted using people from outside the team, external servers whose loads may be extremely difficult to estimate since we do not have data on their assignments outside of this project. The statistical model for the pre-meeting interval shows that having at least one reviewer who is not part of the development team significantly increases the interval. This suggests that external reviewers may handle incoming tasks with a different priority scheme since their main coding tasks are not dependent on the early completion of the inspections.

8 CONCLUSIONS

Previously, we conducted an industrial experiment to understand the factors driving the costs and benefits of software inspections. In this article we described the continued analysis of that experiment. Our goal was to develop an adequate explanatory model of inspection interval.

The analysis suggests that the process environment explains more variation in interval than just the process structure, process inputs, and process techniques. In particular, we find effects due to non-uniform work priorities, time-varying workloads, and the presence of deadlines.

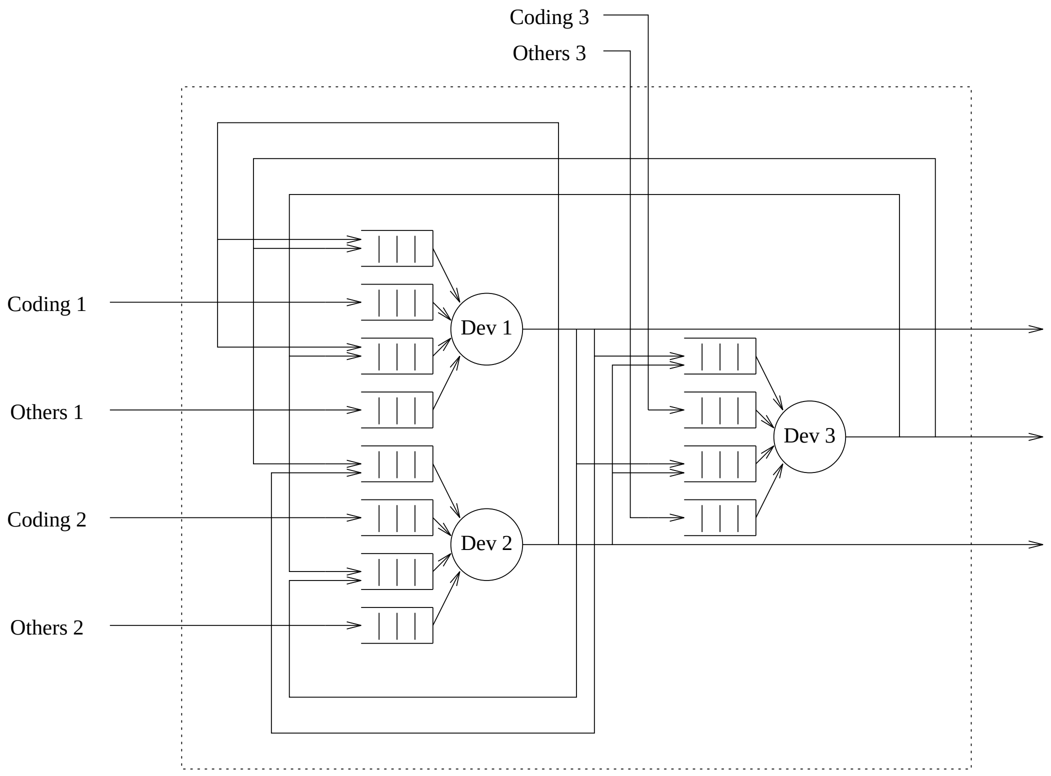

Figure 12: **Global Queueing Model.** This figures depicts a queueing network. The network represents a 3-person development team who interact to develop and inspect software.

We further conjectured that the effects are non-linear and, therefore, linear regression models are inherently inadequate for interval analysis. Instead queueing models may be more appropriate. We also presented a simple example of a queueing model and described some preliminary work to validate them.

8.1 Implications for Software Process Research

One of the advantages of studying software inspection is that they are frequently conducted, they aren't too long in duration, and they share many characteristics of other development processes. Therefore, they are an excellent model for studying the team interaction, communication, scheduling, and analysis found in more general development processes.

Consequently, we believe that many processes besides inspections can benefit from this type of analysis.

8.2 Implications for Practitioners

The ability to model and understand interval has many practical implications. One of the advantages of queueing models is that there is a wide body of literature describing their behavior.

For example, many practitioners have experienced the situation in which a project they thought was near completion dragged on for much longer than they expected. Our results suggest an explanation for some of this behavior. We see that code units enter the system with high priority. However, during the last half of the inspection their priority drops. Queueing analysis tells us that this situation does not minimize sojourn time. These results and others may provide tremendous insight into where bottlenecks exist and what strategies might alleviate them.

References

[1] Richard A. Becker, John M. Chambers, and Allan R. Wilks. *The New S Language.* Wadsworth and Brooks/Cole, 1988.

[2] George E. Box, William G. Hunter, and J. Stuart Hunter. *Statistics for Experimenters.* John Wiley and Sons, Inc., 1978.

[3] Mark G. Bradac, Dewayne E. Perry, and Lawrence G. Votta. Prototyping a process monitoring experiment. In *Proceedings of the 15th International Conference on Software Engineering*, pages 155–165, Baltimore, Maryland, May 1993.

[4] John M. Chambers and Trevor J. Hastie, editors. *Statistical Models in S.* Wadsworth & Brooks, 1992.

[5] Chris Chatfield. Model uncertainty, data mining and statistical inference. *Journal of the Royal Statistical Society, Series A*, 158(3), 1995.

[6] Leonard Kleinrock. *Queueing Systems*, volume II: Computer Applications. John Wiley and Sons, Inc., 1976.

[7] Audris Mockus, Adam A. Porter, Harvey P. Siy, and Lawrence G. Votta. Understanding the sources of variation in software inspections. Technical Report BL0112590-960416-12TM, Bell Laboratories, Lucent Technologies, Naperville, IL, April 1996. Submitted to ACM Trans. on Software Engineering and Methodology.

[8] Adam A. Porter, Lawrence G. Votta, Harvey P. Siy, and Carol A. Toman. An experiment to assess the cost-benefits of code inspections in large scale software development. In *The Third Symposium on the Foundations of Software Engineering*, Washington, D.C., Oct. 1995.

[9] B. Simon. Priority queues with feedback. *Journal of the ACM*, 31(1):134–149, Jan. 1984.

A APPENDIX: STATISTICAL MODELING IN S

A statistical model takes the general form, $y = \mu(X_1) + \epsilon(X_2)$, where y is the vector of observed data, μ is a function taking as input a set X_1 of factors with associated coefficients, $x_{11}, \ldots, x_{1n}$, describing the process and giving as output $\hat{y}$, the expected value of y, and ϵ is a function giving the difference between y and $\hat{y}$, with X_2 being the set of factors, $x_{21}, \ldots, x_{2m}$, in the process whose effects are ignored or whose presence is unknown to us. Model formulation deals mainly with describing μ, specifying factors and the interaction between them. Model fitting deals with moving factors to and from X_1 and X_2 and adjusting the coefficients to give the best fit to y. A model may be considered adequate when ϵ is just white noise, i.e., the residuals $y - \hat{y}$ is a vector of independently distributed values having zero mean and constant variance.

S is a programming environment for data analysis [1, 4]. In this appendix, we will outline our approach in using S to build the statistical models and analyze the data.

A.1 Model Formulation

The possible factors to be incorporated into the model are usually determined from prior knowledge of the process being modeled. The initial model is normally specified with the full set of available factors.

Note that factors may also depend on each other, i.e., have interactions with each other. Each set of possibly interacting factors is represented as an additional factor. (Since we had a limited number of observations, we avoided fitting interaction between factors.)

S has a function, `lm()` for specifying a linear regression model. It takes as basic parameters a model formula[8] and the data for the model.

[8] A model formula is a notation for the structural part of the model, the variable being modeled as well as the factors to explain it. For example, the model formula $y \sim a + b + c$ is read as, "y is modeled by a, b, and c."

A.2 Model Fitting

Model fitting is done by iteratively adding or dropping factors and using regression to adjust the coefficients to give the best fit with the given data. In each iteration, a new factor is added to the model if it significantly reduces the residual variance. Conversely, a factor may be dropped if its removal does not significantly increase the residual variance.

While it is desirable to add as many explanatory factors in the model, there is the danger of adding too many factors. This is known as overfitting [5]. The problem is that while the model might be a good fit to the data it is modeled on, it may be inexplainable or may not make physical sense. In addition, it cannot reliably characterize and predict additional data. One way to check this is to partition the data and build the model on one set and test its reliability on the other set. However, as in our case, there are usually too few data points to begin with.

We looked for a parsimonious model with the help of stepwise model selection. In stepwise model selection, we start with an existing model and iteratively add or drop one term, minimizing the number of parameters while maximizing the fit according to some specified criterion. In S, we used the function `step()`, increasing the scale parameter until the number of factors in the model are sufficiently reduced.

The model selection algorithm may not give the best model since it does not know the physical meaning of the factors it manipulates. At the end, we must use our prior knowledge of the process in order to fine-tune the model to one that is physically interpretable.

A.2.1 Calculating the Significance

To calculate the significance of a factor's contribution into the model, we used the `summary.aov()` function to perform analysis of variance, passing the model specification into it, with the factor of interest at the end of the formula. For example, if we have a model $y \sim a + b + c$, we perform the analysis of variance on $y \sim b + c + a$, $y \sim a + c + b$, and $y \sim a + b + c$ to calculate the significance of the contributions of a, b, and c to the model. Essentially, this is how `step()` determines which factor to retain and which to drop.

A.3 Model Checking

Once a model has been specified and fitted, it is checked to see if it is an adequate model. The model is adequate when it reasonably estimates y, i.e., there is a high linear correlation between y and $\hat{y}$, and has sufficiently explained the variance, i.e., the residuals are reduced to a patternless set of data as plotted against y and against $\hat{y}$.