The Impact of Design and Code Reviews on Software Quality: An Empirical Study Based on PSP Data

Chris F. Kemerer, *Member*, *IEEE Computer Society*, and Mark C. Paulk, *Senior Member*, *IEEE*

Abstract—This research investigates the effect of review rate on defect removal effectiveness and the quality of software products, while controlling for a number of potential confounding factors. Two data sets of 371 and 246 programs, respectively, from a Personal Software Process (PSP) approach were analyzed using both regression and mixed models. Review activities in the PSP process are those steps performed by the developer in a traditional inspection process. The results show that the PSP review rate is a significant factor affecting defect removal effectiveness, even after accounting for developer ability and other significant process variables. The recommended review rate of 200 LOC/hour or less was found to be an effective rate for individual reviews, identifying nearly two-thirds of the defects in design reviews and more than half of the defects in code reviews.

Index Terms—Code reviews, design reviews, inspections, software process, software quality, defects, software measurement, mixed models, personal software process (PSP).

1 Introduction

Quality is well understood to be an important factor in software. Deming succinctly describes the business chain reaction resulting from quality: improving quality leads to decreasing rework, costs, and schedules, which all lead to improved capability, which leads to lower prices and larger market share, which leads to increased profits and business continuity [14]. Software process improvement is inspired by this chain reaction and focuses on implementing disciplined processes, i.e., performing work consistently according to documented policies and procedures [37]. If these disciplined processes conform to accepted best practice for doing the work, and if they are continually and measurably improving, they are characterized as mature processes.

The empirical evidence for the effectiveness of process improvement is typically based on before-and-after analyses, yet the quality of process outputs depends upon a variety of factors, including the objectives and constraints for the process, the quality of incoming materials, the ability of the people doing the work, and the capability of the tools used, as well as the process steps followed. Empirical analyses are rarely able to control for differences in these factors in real-world industrial projects. And, even within such a project, these factors may change over the project's life.

- *C.F. Kemerer is with the Katz Graduate School of Business, University of Pittsburgh, 278A Mervis Hall, Pittsburgh, PA 15260. E-mail: ckemerer@katz.pitt.edu.*
- *M.C. Paulk is with the IT Services Qualification Center, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213. E-mail: mcp@cs.cmu.edu.*

Manuscript received 12 Nov. 2007; revised 6 Jan. 2009; accepted 13 Jan. 2009; published online 1 Apr. 2009.
Recommended for acceptance by A.A. Porter.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number TSE-2007-11-0322.
Digital Object Identifier no. 10.1109/TSE.2009.27.

An example of an important process where there is debate over the factors that materially affect performance is the inspection of work products to identify and remove defects [17], [18]. Although there is general agreement that inspections are a powerful software engineering technique for building high-quality software products [1], Porter and Votta's research concluded that "we have yet to identify the fundamental drivers of inspection costs and benefits" [44]. In particular, the optimal rate at which reviewers should perform inspections has been widely discussed, but subject to only limited investigations [6], [21], [45], [47].

The research reported in this paper investigates the impact of the review rate on software quality, while controlling for a comprehensive set of factors that may affect the analysis. The data come from the Personal Software Process (PSP), which implements the developer subset of the activities performed in inspections. Specifically, the PSP design and code review rates correspond to the preparation rates in inspections.

The paper is organized as follows: Section 2 describes relevant previously published research. Section 3 describes the methodology and data used in the empirical analyses. Section 4 summarizes the results of the various statistical models characterizing software quality. Section 5 describes the implications of these results and the conclusions that may be drawn.

2 Background

Although a wide variety of detailed software process models exists, the software process can be seen at a high level as consisting of activities for requirements analysis, design, coding, and testing. Reviews of documents and artifacts, including design documents and code, are important quality control activities, and are techniques that

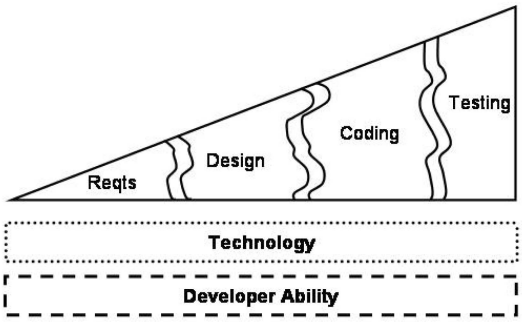

Fig. 1. A conceptual view of the software development process and its foundations.

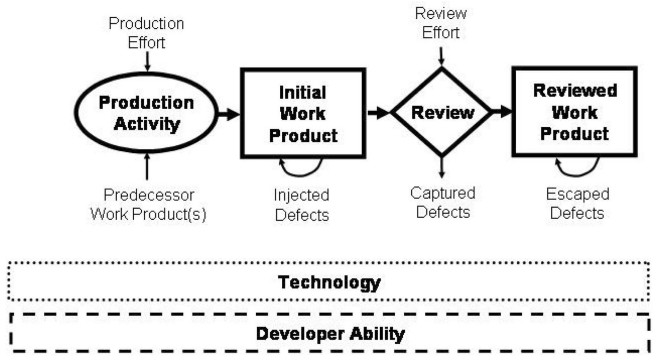

Fig. 2. Production and review steps.

can be employed in a wide variety of software process life cycles [36]. Our analysis of the impact of review rate on software quality is based on previous empirical research on reviews and it considers variables found to be useful in a number of defect prediction models [5], [32].

2.1 A Life Cycle Model for Software Quality

A conceptual model for the software life cycle is illustrated in Fig. 1. It shows four primary engineering processes for developing software—requirements analysis of customer needs, designing the software system, writing code, and testing the software. A process can be defined as a set of activities that transforms inputs to outputs to achieve a given purpose [36]. As illustrated in Fig. 1, the engineering processes within the overall software life cycle transform input work products, e.g., the design, into outputs, e.g., the code, which ultimately result in a software product delivered to a customer. These general engineering processes may be delivered via a variety of life cycles, e.g., evolutionary or incremental. The quality of the outputs of these engineering processes depends on the ability of the software professionals doing the work, the activities they do, the technologies they use, and the quality of the input work products.

Early empirical research on software quality identified size as a critical driver [2], and the size of work products remains a widely used variable in defect prediction models [32]. Customer requirements and the technologies employed (such as the programming language) are primary drivers of size. Finally, since software development is a human-centric activity, developer ability is commonly accepted as a critical driver of quality [3], [12], [13], [49], [55].

The cumulative impact of the input quality can be seen in the defect prevention models that use data from the various phases in the software's life cycle to estimate test defects. For example, predicting the number of released defects has been accomplished by multiplying the sizes of interim work products by the quality of the work products and the percentage of defects escaping detection [11].

2.2 Production of Engineering Work Products

Although software quality can be characterized in a number of ways, defects, as measured by *defect density* (defects/lines of code), are a commonly used quality measure [22], [52]. Typical reported defect density rates range from 52 to 110 per thousand lines of code (KLOC) [5], [24].[1]

Empirical research in software defect prediction models has produced a range of factors as drivers for software quality. As different models have been found to be the best for different environments, it appears unlikely that a single superior model will be found [8], [53]. For example, Fenton and Neil point out that defect prediction models based on measures of size and complexity do not consider the difficulty of the problem, the complexity of the proposed solution, the skill of the developer, or the software engineering techniques used [19]. Therefore, the extent to which these factors (and potentially others) can affect software quality remains an open empirical question. However, based on the prior literature and starting from the general model shown in Fig. 1, the relevant software engineering activity can be described in terms of a *production* step and a *review* step, as shown in Fig. 2.

This figure can be read from left to right as follows. Production results in an initial work product whose quality in terms of injected defects depends upon the quality of predecessor work products, the technologies used in production, the ability of the developer, and the effort expended on production. The quality of production can be measured by the number of defects in the resulting work product, which typically is normalized by the size of the work product to create a defect density ratio [5], [22], [24], [52]. The initial work product may be reviewed to capture and remove defects, and the quality of the resulting corrected work product depends upon the size and quality of the initial work product, the ability of the developer, and the effort expended in the review. Given a measure of the number of defects in the work product at the time of the review, the quality of reviews can be seen as the effectiveness of the review in removing defects.

2.3 Reviews

Reviews of work products are designed to identify defects and product improvement opportunities [36]. They may be performed at multiple points during development, as

1. It should be noted that a complete view of "quality" would include many attributes, such as availability, features, and cost. However, as an in-process measure, the number of defects in the software provides insight into potential customer satisfaction, when the software will be ready to release, how effective and efficient the quality control processes are, how much rework needs to be done, and what processes need to be improved. Defects are, therefore, a useful, if imperfect, measure of quality.

opposed to testing, which typically can occur only after an executable software module is created. A crucial point in understanding the potential value of reviews is that it has been estimated that defects escaping from one phase of the life cycle to another can cost an order of magnitude more to repair in the next phase, e.g., it has been estimated that a requirements defect that escapes to the customer can cost 100-200 times as much to repair as it would have cost if it had been detected during the requirements analysis phase [5]. Reviews, therefore, can have a significant impact on the cost, quality, and development time of the software since they can be performed early in the development cycle.

It is generally accepted that inspections are the most effective review technique [1], [21], [23], [47]. A typical set of inspection rules includes items such as:

- The optimum number of inspectors is four.
- The preparation rate for each participant when inspecting design documents should be about 100 lines of text/hour and no more than 200 lines of text/hour.
- The meeting review rate for the inspection team in design inspections should be about 140 lines of text/hour and no more than 280 lines of text/hour.
- The preparation rate for each participant when inspecting code should be about 100 LOC/hour and no more than 200 LOC/hour.
- The meeting review rate for the inspection team in code inspections should be about 125 LOC/hour and no more than 250 LOC/hour for code.
- Inspection meetings should not last more than two hours.

Many variant inspection techniques have been proposed. Gilb and Graham, for example, suggest a preparation rate of 0.5 to 1.5 pages per hour; they also suggest that rates as slow as 0.1 page per hour may be profitable for critical documents [21]. The defect removal effectiveness reported for different peer review techniques (e.g., inspections, walk-throughs, and desk checks) ranges from 30 to over 90 percent, with inspections by trained teams beginning at around 60 percent and improving as the team gains experience [17], [18], [33], [47].

Despite consistent findings that inspections are generally effective, Glass has summarized the contradictory empirical results surrounding the factors that *lead* to effective inspections [23]. Weller found that the preparation rate for an inspection, along with familiarity with the software product, were the two most important factors affecting inspection effectiveness [50]. Parnas and Weiss argue that a face-to-face meeting is ineffective and unnecessary [35]. Eick et al. found that 90 percent of the defects could be identified in preparation, and therefore, that face-to-face meetings had negligible value in finding defects [16]. Porter and his colleagues created a taxonomy of review factors that they argue should be empirically explored [44], [45], including:

- structure, e.g., team size, the number of review teams, and the coordination strategy for multiple teams [42];
- techniques, e.g., individual versus cooperative reviews; ad hoc, checklist-based, and scenario-based [30], [40], [43];
- inputs, e.g., code size, functionality of the work product, the producer of the work product, and the reviewers [45];
- context, e.g., workload, priorities, and deadlines [41]; and
- technology, e.g., Web-based workflow tools [39].

In summary, although the benefits of inspections are widely acknowledged, based on these competing views and conflicting arguments the discipline has yet to fully understand the fundamental drivers of inspection costs and benefits [23], [44].

2.4 The Personal Software Process (PSP)

In order to address the empirical issues surrounding the drivers for effective inspections, it will be beneficial to focus on specific factors in a bounded context. The PSP incrementally applies process discipline and quantitative management to the work of the individual software professional [25]. As outlined in Table 1, there are four PSP major processes (PSP0, PSP1, PSP2, and PSP3) and three minor extensions to those processes. Each process builds on the prior process by adding engineering or management activities. Incrementally adding techniques allows the developer to analyze the impact of the new techniques on his or her individual performance. The life cycle stages for PSP assignments include planning, design, coding, compiling, testing, and a postmortem activity for learning, but the primary development processes are design and coding, since there is no requirements analysis step. When PSP is taught as a course, there are 10 standard assignments and these are mapped to the four major PSP processes in Table 1.

Because PSP implements well-defined and thoroughly instrumented processes, data from PSP classes are frequently used for empirical research [20], [24], [46], [51], [54]. PSP data are well suited for use in research as many of the factors perturbing project performance and adding "noise" to research data, such as requirements volatility and teamwork issues, are either controlled for or eliminated in PSP. And, since the engineering techniques adopted in PSP include design and code reviews, attributes of those reviews affecting individual performance can be investigated.

In PSP, a defect is a flaw in a system or system component that causes the system or component to fail to perform its required function. While defects in other contexts may be categorized according to their expected severity, in PSP, defects are not "cosmetic," i.e., a PSP defect, if encountered during execution, will cause a failure of the system.

Researchers Hayes and Over observed a decrease in defect density as increasingly sophisticated PSP processes were adopted, along with improvements in estimation accuracy and process yield [24]. Their study was replicated by Wesslen [51]. Wohlin and Wesslen observed that both the average defect density and the standard deviation decreased across PSP assignments [54]. Prechelt and Unger observed fewer mistakes and less variability in performance as PSP assignments progressed [46]. In a study of three improvement programs, Ferguson et al. observed that PSP accelerated organizational improvement efforts (including improved planning and scheduling), reduced development

TABLE 1
Description of the PSP Processes and Assignments

PSP Process	Process Description and Related Assignments
PSP0	The "current" process of the developer at the beginning of the course. Basic measures of historical size, time, and defect data are collected to establish an initial baseline. Assignment 1A.
PSP0.1	Adds a coding standard, process improvement proposals, and size measurement. Assignments 2A and 3A.
PSP1	Adds size estimating and test reports. Assignment 4A.
PSP1.1	Adds task planning and schedule planning. Assignments 5A and 6A.
PSP2	Introduces design reviews and code reviews – personal reviews conducted by an engineer on his or her own design or code to remove all defects before compiling the program. Assignments 7A and 8A.
PSP2.1	Adds design templates for functional specifications, state specifications, logic specifications, and operational scenarios. Assignment 9A.
PSP3	Introduces the concept of cyclic development – incrementally building a program in multiple cycles. Assignment 10A.

time, and resulted in better software [20]. Using PSP data, Paulk found that developer ability reinforced the consistent performance of recommended practices for improved software quality [38].

3 Methodology

PSP employs a reasonably comprehensive and detailed set of process and product measures, which provide a rich data set that can be statistically analyzed in order to estimate the effect of process factors, while controlling for technology and developer ability inputs. However, quality factors associated with volatile customer requirements, idiosyncratic project dynamics, and ad hoc team interactions are eliminated in the PSP context. The reviews in PSP correspond to the checklist-based inspections described by Fagan, but PSP reviews are performed by the developer only; no peers are involved. Therefore, review rates in PSP correspond to the developer preparation rates in inspections.

Since the PSP data provide insight into a subset of the factors that may be important for peer reviews in general, this analysis provides a "floor" for inspection performance in the team environment. Of course, a key benefit of analyzing a subset of the factors is that we can isolate specific factors of interest. These results can then be considered a conservative estimate of performance as additional elements, such as team-based reviews, and variant elements, such as scenario-based reading techniques, are possibly added. In addition, this research investigates the contributors to quality in the PSP processes at a finer level of granularity than has been performed in typical prior empirical PSP analyses.

3.1 The PSP Data Sets

This research uses PSP data from a series of classes taught by Software Engineering Institute (SEI) authorized instructors. Since the focus of this research is on review effectiveness, only data from the assignments following the introduction of design and code reviews, i.e., assignments 7A to 10A, are used. These correspond to the final two PSP processes, numbers 3 and 4, and represent the most challenging assignments.

Johnson and Disney have identified a number of potential research concerns for PSP data validity centered on the manual reporting of personal data by the developers, and they found about 5 percent of PSP data in their study to be unreliable [28]. For our PSP data set, the data were checked to identify any inconsistencies between the total number of defects injected and removed, and to identify instances where the number of defects found in design review, code review, or compile exceeded the number reported to have been injected at that point, suggesting that one count or the other was in error. As a result of this consistency check, 2.9 percent of the data was excluded from the original set of observations. This rate is similar to Johnson and Disney's rate, and the smaller degree of invalid data may be attributed to the fact that some of the classes of data errors they identified, such as developer calculation errors, would not be present in our study because we use only the reported base measures and none of the analyses performed by the PSP developers. Although data entry errors can be a concern in any empirical study, since they occur in the data collection stage and are difficult to identify and correct, fewer than 10 percent of the errors identified by Johnson and Disney (or less than 0.5 percent of their data) were entry errors. There is no reason to believe that such data entry errors are more likely in our PSP data set, nor that such errors are likely to be distributed in a manner that would bias the results.

To control for the possible effect of the programming language used, the PSP data set was restricted to assignments

TABLE 2
Variable Names and Definitions for the Models

Category	Variable Name	Definition	Units of Measure			
			KLOC	(-) defects / KLOC	hours / KLOC	% of defects identified
Control variable	Size	Program size as measured by thousands of lines of code (KLOC)	√			
Control variable	Ability	Developer ability (as measured by average defect density in testing for assignments 1A-3A, prior to reviews)		√		
Design	InitDsnQual	Initial design quality (before design review)		√		
Design	DsnRevRate	Design review rate			√	
Design	DsnRevEffect	Defect removal effectiveness of design review				√
Design	DsnQual	Design quality (after design review)		√		
Coding	InitCodeQual	Initial code quality (before code review)		√		
Coding	CodeRevRate	Code review rate			√	
Coding	CodeRevEffect	Defect removal effectiveness of code review				√
Coding	CodeQual	Code quality (after code review)		√		
Testing	TestQual	Software quality in test		√		

done in either C or C++ (the most commonly used PSP programming languages), and the data for each language were analyzed separately. Since the focus of the analyses is on the impact of design and code reviews, only those assignments where reviews occurred were included. In addition, some observations had no recorded defects at the time of the review. As no factor can affect review effectiveness if no defects are present, these reviews were also excluded from the data set. After these adjustments, the resulting C data set has 371 observations for 153 developers and the C++ data set has 246 observations for 90 developers. Note that a single developer may have up to four observations in a data set, which is one driver for the mixed models analysis later in this paper.

3.2 The Basic Models

Our basic quality model is derived in two parts: a high-level model ("project level") and a lower level of detail model ("engineering activity level"). Ultimately, a customer cares about the quality of the software as delivered at the end of the project, and therefore, the high-level project objective of reliably addressing customer needs is fundamental to effective software process improvement. The process as conceptually captured in Fig. 1 for the project as a whole can be modeled as follows:

$$Software\ quality = f\ (Developer\ ability,\ Technology,\ Requirements\ quality,\ Design\ quality,\ Code\ quality).$$

A cutoff point is needed for counting and comparing the total number of defects in a product. For PSP, acceptance of a work product (assignment) by the instructor constitutes the most logical cutoff point; therefore, software quality is defined for purposes of this model as defect density measured in testing.

The quality of work products, such as requirements, design, and code (as depicted in Fig. 2) can be modeled as a function of the initially developed work product quality and the effectiveness of the review. As the focus of this research is on the effectiveness of the reviews, this leads to the following model for the defect removal effectiveness of a review as performed by an individual software engineer:

$$Review\ effectiveness = f\ (Developer\ ability,\ Technology,\ Review\ rate).$$

Review rate (effort/size) is both a management control and likely to be a driver of review effectiveness within the context of developer ability and technology [17], [47]. While other factors could be considered for *team-based* peer reviews, the factors shown here are believed to be the critical factors for PSP's checklist-based *individual* reviews.

3.3 Operationalizing the Variables

The operational definitions of the variables used in these models are contained in Table 2. The processes characterized by these variables are design, code, and test. In the context of a PSP assignment, the requirements can be considered a defect-free work product, given that they are identical for each developer. For purposes of the research, this controls for variance that might otherwise be attributable to defects or changes in requirements, and allows for the analysis of variance attributable statistically to the review process. Design and code quality can be determined in operational terms since causal analysis performed by the developer, who is closest to the work, informs us of how

TABLE 3
Range of Important PSP Data Values

Variable	C Data Set			C++ Data Set		
N =	371			246		
	minimum	median	maximum	minimum	median	maximum
Lines of code	17	120	985	18	124	913
Number of design defects	1	3	34	1	3	55
Design review time (min)	1	21	180	1	23	131
Design review rate (LOC/hr)	27	296	4740	32	338	5580
Number of code defects	1	6	45	1	7	56
Code review time (min)	2	26	227	4	28	145
Code review rate (LOC/hr)	41	261	2463	30	277	1896
Number of test defects	0	1	16	0	1	26

many defects were injected and removed at each step in the PSP process.

Starting at the top of Table 2, the *Size* of the system in lines of code is a commonly used measure. *Ability* is measured by averaging the defect density in testing for the first three assignments. Developers are, therefore, rated on a set of tasks that is similar to the ones that will follow, and in the same setting. We would expect that developers who do well on the first three tasks are likely to do well on those that follow, *ceteris paribus*. It should be noted that since the first three assignments do not include reviews, this is a measure of developer ability and does not specifically include reviewer ability. The measure of defect density, (defects/KLOC) is then inverted (-defects/KLOC) to create the measure of ability so that larger values can be intuitively interpreted as "better."

Similarly, all of the quality variables (*InitDsnQual, DesignQual, InitCodeQual, CodeQual*, and *TestQual*) are measured using this same unit directionality (-defects/KLOC), although the measurements are taken at different points in the process.

Review rates combine program size, expressed in KLOC, and effort, expressed in hours. When review rates are measured as the ratio of hours/KLOC, there is a natural, increasing progression from fast reviews (those with a relatively small number of hours spent) to slower reviews. For example, a recommended rate of less than 200 LOC/hour is transformed to a value greater than 5 hours/KLOC, and in this manner, larger values can intuitively be interpreted as "better." The two review rates (*DesnRevRate* and *CodeRevRate*) are measured as hours/KLOC.

When a defect is identified in PSP, causal analysis by the developer identifies where in the life cycle the defect was injected. The number of defects in the design and code at the end of each PSP step can, therefore, be determined based on the number of defects injected or removed during the step. The total number of defects for a PSP program is operationally defined as the number identified by the end of testing when the assignment is accepted. Design and code review effectiveness (*DsnRevEffect* and *CodeRevEffect*) are measured as the percentage of defects in the design and code, respectively, which were identified in their reviews. Finally, technology can be controlled as a factor by splitting the data set into subsets by programming language.

To give a feel for these PSP programs, the ranges of the important numbers that these variables are based on are provided in Table 3. The minimum, median, and maximum values are listed.

As discussed in Section 3.1, observations with zero defects at the time of design review or code review were removed, therefore, the minimums must be at least one for these two variables. This is not the case for the number of test defects. In addition to the detailed statistics in Table 3, we also provide sample box plots for the C data set, the larger of the two, in order to provide some intuitive feel for the data. As can be observed in Fig. 3 for lines of code and Fig. 4 for the number of

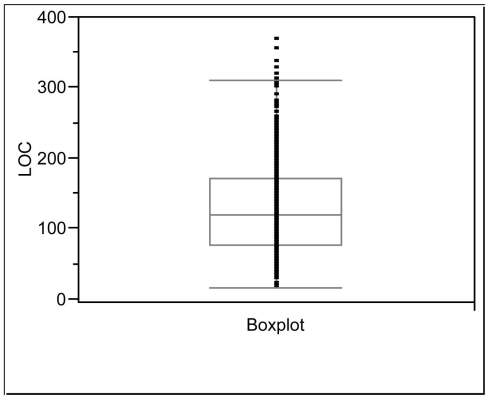

Fig. 3. Box and whisker chart for lines of code: C data set.

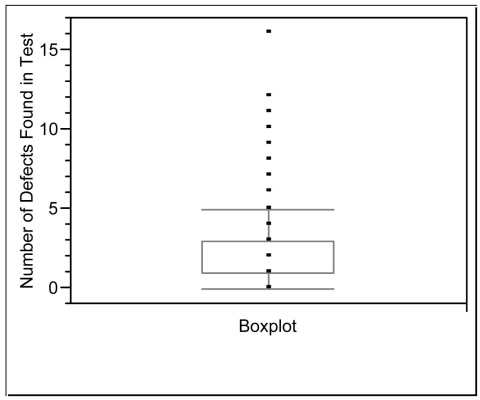

Fig. 4. Box and whisker chart for number of test defects: C data set.

TABLE 4
Regression Results for the Life Cycle Models

	C Data Set	C++ Data Set
Prob > F	<0.0001	<0.0001
R^2_a	0.5845	0.6513
Coefficient estimates (standard errors)		
β_0	-0.12 (1.09)	2.47 (1.73)
Ability	0.05**** (0.02)	0.0002 (0.03)
DsnQual	0.10*** (0.03)	0.10** (0.05)
CodeQual	0.38**** (0.02)	0.46**** (0.03)

*$p<0.10$; **$p<0.05$; ***$p<0.01$; ****$p<0.001$

test defects, these data sets have the typical skewness seen in most empirical data sets, and therefore, we will employ an analysis that takes the presence of outliers into account.

4 Statistical Modeling

We begin with an analysis using multiple regression models that allow us to examine the impact on quality of process variables, such as review rate, in a context where other effects, such as developer ability and the programming language used, can be controlled.[2] Splitting the data sets by programming language, in addition to addressing the potential impact of technology differences, also allows us to replicate our analyses.

4.1 Life Cycle Models for Software Quality

The project life cycle model in Fig. 1 expressed as a regression model for quality is

$$TestQual = \beta_0 + \beta_1\ (Ability) + \beta_2\ (DsnQual) + \beta_3\ (CodeQual) + \varepsilon. \quad (1)$$

The quality of the product depends upon the ability of the developer and the quality of the predecessor work products. We expect that the quality of the software as measured in test (*TestQual*) will increase as developer ability grows (β_1 is expected to be positive) and as the quality of the design and code improve (β_2 and β_3 are expected to be positive), where β_0 is the intercept term and ε is the error term.

This model was estimated for both the C and C++ data sets as described in Table 4. Both models are statistically significant, accounting for approximately 58 and 65 percent of the variation in the data, respectively, as measured by the adjusted r-squared statistic R^2_a.

We would expect better developers to do better, all else being equal, so we control for that effect by including the *Ability* variable in the model. As indicated by the positive coefficients, the data support this interpretation. (Note that the coefficient for ability in the C++ model, while positive, is not significantly different from zero at usual statistical levels.) Having included this variable in the model, we can now interpret the coefficients of the other variables with greater confidence, since variation that should properly be attributed to the developers has been accounted for.

As expected, the quality of the predecessor work products was found to be statistically significant for both data sets, with the quality of the most recent work product, the code, having the larger impact on software quality. The positive sign for the coefficient estimates indicates that as the quality of the design and code improve, so does the quality of the software as measured in testing, as expected.

The correlation matrix for the independent variables in the life cycle models is shown in Table 5. The relatively high correlations for design and code quality may suggest that good designers also tend to be good coders all else being equal.

Collinearity diagnostics indicate that collinearity is not a significant issue for these models. The condition numbers for the two models range from 4.1 to 4.4, which is less than Belsley et al.'s suggested limit of 20 [4]. The maximum variance inflation factors (VIFs) for the predictor variables range from 1.1 to 1.8, which is less than the suggested maximum of 10 [34].

A number of standard diagnostics were used to identify influential outliers that might unduly affect a regression model: leverage, studentized deleted residual, Cook's distance, and DFFITS [34]. For the C data set, 17 influential outliers were identified by at least one of these diagnostics; for the C++ data set, 15 influential outliers were identified. Models for the data sets excluding influential outliers, which are shown in Table 6, were not materially different from those including the outliers, although R^2_a increased to 71 and 75 percent, respectively, and the sum of squares of the errors (*SSE*) decreased 33 and 48 percent, respectively. Therefore, the managerial conclusions from the model excluding influential variables are not materially different from those for the model with all of the data points.

2. Due to the occurrence of multiple observations per developer, the preferred analysis technique is *mixed models*, which we present in Section 4.3. However, we begin with an analysis using ordinary least squares regression as the interpretation of the results will be more familiar to *TSE* readers, and this method can be relatively robust to some specification errors. (In fact, the results from Sections 4.2 and 4.3 are very similar.)

TABLE 5
Correlation Matrix for the Independent Variables in the Life Cycle Models

	C Data Set			C++ Data Set		
	Ability	**DsnQual**	**CodeQual**	**Ability**	**DsnQual**	**CodeQual**
Ability	--	0.2019****	0.2264****	--	0.1547**	0.2304****
DsnQual	--	--	0.4132****	--	--	0.6427****
CodeQual	--	--	--	--	--	--

*$p<0.10$; **$p<0.05$; ***$p<0.01$; ****$p<0.001$

TABLE 6
Life Cycle Models Excluding Outliers and with Transformations

	C Data Set			C++ Data Set		
	Excluding Outliers	**Box Cox Transforms of Y**	**Ln Transforms of X_i**	**Excluding Outliers**	**Box Cox Transforms of Y**	**Ln Transforms of X_i**
Prob > F	<0.0001	<0.0001	<0.0001	<0.0001	<0.0001	<0.0001
R^2_a	0.7115	0.6289	0.5606	0.7546	0.6051	0.5441
Coefficient estimates (standard errors)						
β_0	2.47**** (0.95)	4.90**** (0.80)	-16.22**** (2.53)	4.09*** (1.39)	6.13**** (0.97)	-6.07** (2.91)
Ability	0.05**** (0.01)	-0.04**** (0.01)	2.35**** (0.65)	0.02 (0.03)	-0.02 (0.02)	-0.51 (0.79)
DsnQual	0.11**** (0.03)	-0.06** (0.02)	0.98*** (0.32)	0.31**** (0.04)	-0.14**** (0.03)	2.54**** (0.40)
CodeQual	0.49**** (0.02)	-0.34**** (0.02)	7.84**** (0.43)	0.43**** (0.03)	-0.22**** (0.02)	6.22**** (0.52)

*$p<0.10$; **$p<0.05$; ***$p<0.01$; ****$p<0.001$

Transformations of the dependent variable attempt to provide a simple, normal linear model that simultaneously satisfies constant variance, normality, and $E(\hat{Y}) = X\beta$ [48]. The life cycle models are described in Table 6, with a Box Cox transformation of the dependent variable (without influential outliers) and with a natural logarithm transformation of the independent variables (without influential outliers and using the Box Cox transformation of the dependent variable).[3] The Box Cox transformation decreased *SSE* 30 percent for the C++ data set with $\lambda = 0.5$ and 51 percent for the C data set with $\lambda = 0.4$) [34]. A natural logarithm (Ln) transformation of the independent variables was also investigated, but was not found to add explanatory power. Although heteroscedasticity was identified by White's test for both models in Table 4, it is not an issue for any of the transformed models in Table 6. The managerial conclusions using the Box Cox transform of the independent variable or the natural logarithm transformations of the dependent variables are not materially different from those for the initial simpler model form.[4] Finally, interaction effects were also investigated, but these more complex life cycle models did not provide additional managerial insight [38].

In summary, both higher design quality and higher code quality are consistently associated with higher software quality as measured by fewer test defects in both data sets. As was the case with the simpler linear model, greater developer ability was also significantly associated with higher software quality for the C data set only. Overall, these results support the general intuition that developer ability and input quality are expected to be related to output quality. Having established that these expected relationships hold for our data, in the next section we turn to our primary research questions, which are the effect of design and code review rates on quality, while controlling for ability and input quality.

4.2 Defect Removal Effectiveness Models

Having empirically verified in Section 4.1, the intuition from Fig. 1 that software quality depends upon the quality of the predecessor work products used to build the software and on developer ability, the next step is to analyze the factors affecting the quality of those predecessor work products. As modeled in Fig. 2, quality depends upon initial work product quality and review effectiveness. It is an open question as to whether there should be a relationship between initial work product quality and defect removal effectiveness since ability as both a developer and a reviewer need not coexist. Therefore, we include initial work product quality to allow

3. Note that, when making a Box Cox transformation, values >0 are desired for the dependent variable and a small offset was added to prevent an undefined Ln(0) transformation. As a result, the expected coefficients for the ability and quality variables are negative for the Box Cox and Ln transformations.

4. Note that the sign of the coefficient for the Box Cox transform changes due to the necessity of using an unadjusted (positive) ratio for defect density, rather than the more managerially intuitive negative ratio.

TABLE 7
Regressions for Defect Removal Effectiveness

	Design			Code	
	C Data Set	C++ Data Set		C Data Set	C++ Data Set
Prob > F	0.0072	0.0064	**Prob > F**	<0.0001	<0.0001
R^2_a	0.0244	0.0376	R^2_a	0.0673	0.1442
Coefficient estimates (standard errors)					
β_0	0.58**** (0.04)	0.50**** (0.05)	β_0	0.46**** (0.03)	0.47**** (0.03)
Ability	0.001** (0.0005)	0.0001 (0.0007)	**Ability**	0.0004 (0.0004)	0.0006 (0.0005)
InitDsnQual	0.001 (0.0007)	0.002*** (0.0007)	**InitCodeQual**	0.0008** (0.0003)	0.001**** (0.0003)
DRRate	0.01** (0.005)	0.02**** (0.007)	**CRRate**	0.03**** (0.005)	0.03**** (0.005)

*$p<0.10$; **$p<0.05$; ***$p<0.01$; ****$p<0.001$

TABLE 8
Correlation Matrix for the Independent Variables in the Defect Removal Effectiveness Models

	Design Reviews					
	C Data Set			C++ Data Set		
	Ability	**InitDsnQual**	**DRRate**	**Ability**	**InitDsnQual**	**DRRate**
Ability	--	0.1354	-0.1416	--	0.1678***	-0.1509**
InitDsnQual	--	--	-0.2912	--	--	-0.5638****
DRRate	--	--	--	--	--	--
	Code Reviews					
	C Data Set			C++ Data Set		
	Ability	**InitCodeQual**	**CRRate**	**Ability**	**InitCodeQual**	**CRRate**
Ability	--	0.2590****	-0.1137**	--	0.2053***	-0.1303**
InitCodeQual	--	--	-0.4553****	--	--	-0.7088****
CRRate	--	--	--	--	--	--

*$p<0.10$; **$p<0.05$; ***$p<0.01$; ****$p<0.001$

us to investigate the issue. We expect that the defect removal effectiveness of a review will increase as developer ability grows, and as more care is taken to do a careful review, note that the review rate increases since it is measured in hours per KLOC in these analyses. We also expect the quality of the work product to improve as its initial quality increases.[5]

The defect removal effectiveness of a review is modeled as a function of developer ability, the quality of the initially developed work product, and the review rate:

$$DsnRevEffect = \beta_0 + \beta_1\,(Ability) + \beta_2\,(InitDsnQual) + \beta_3\,(DsnRevRate). \quad (2)$$

Similarly, the model for defect removal effectiveness of code reviews is

$$CodeRevEffect = \beta_0 + \beta_1\,(Ability) + \beta_2\,(InitCodeQual) + \beta_3\,(CodeRevRate). \quad (3)$$

These two models were estimated for both the C and C++ data sets as described in Table 7.

All four models are statistically significant at usual levels. After the addition of the Design Review Rate (DRRate) variable, Ability and Initial Design Quality continue to have positive coefficients, although some of these coefficients are not significantly different from zero at usual levels of statistical significance. DRRate (hours/KLOC) is consistently and positively associated with defect removal effectiveness at statistically significant levels in all four cases, i.e., for both data sets in both design and code, even after controlling for developer ability and the initial work product quality.

The correlation matrix for the independent variables in this linear model is presented in Table 8. The relatively high negative correlations between the initial quality of a work product and the review rate suggest that developers who build high-quality work products may also take less time reviewing those work products. The positive correlation of developer ability with initial work product quality, along with the negative correlation with review rate, suggests that reviewer ability may be relatively independent of developer ability.

5. Note that, although removing a single defect has a greater percentage impact on effectiveness, when there are a smaller number, such defects will be correspondingly harder to find.

TABLE 9
Design Review Effectiveness Models, Excluding Outliers, with Transformations

	C Data Set			C++ Data Set		
	Excluding Outliers	**Ln Transforms of X_i**	**Logit Transforms of Y**	**Excluding Outliers**	**Ln Transforms of X_i**	**Logit Transforms of Y**
Prob > F	0.0048	<0.0001	<0.0001	0.0040	<0.0001	<0.0001
R^2_a	0.0268	0.0867	0.1806	0.0422	0.0821	0.1181
	Coefficient estimates (standard errors)					
β_0	0.58**** (0.04)	0.90**** (0.12)	1121.34**** (136.36)	0.53**** (0.05)	0.64**** (0.13)	838.10**** (152.90)
Ability	0.001*** (0.0005)	-0.07** (0.03)	-46.78 (30.15)	0.0004 (0.0008)	0.006 (0.03)	-23.20 (38.28)
InitDsnQual	0.001 (0.0007)	-0.09*** (0.03)	-262.94**** (29.98)	0.002*** (0.0008)	-0.10*** (0.03)	-192.23**** (33.51)
DRRate	0.01** (0.005)	0.15**** (0.03)	154.90**** (30.17)	0.02*** (0.007)	0.15**** (0.03)	102.18*** (35.18)

*$p<0.10$; **$p<0.05$; ***$p<0.01$; ****$p<0.001$

Collinearity diagnostics indicate that collinearity is not a significant issue for these models. The condition numbers for the various models range from 4.2 to 5.0, which is less than the suggested limit of 20 [4]. The maximum VIF for the predictor variables ranges from 1.0 to 2.1, which is less than the suggested maximum of 10 [34]. For design reviews using the C data set, one influential outlier was identified; for the C++ data set, three influential outliers were identified. For code reviews using the C data set, six influential outliers were identified; for the C++ data set, five influential outliers were identified. Although the arcsine root and logit transformations are frequently proposed for percentage data [7], the logit transformation of the dependent variable was found to be the best for this data set. A natural logarithm transformation of the independent variables was found to improve the fit of the model.

Three defect removal effectiveness models for design reviews are described in Table 9 for both the C and C++ data sets: 1) with influential outliers excluded; 2) with influential outliers excluded and a natural logarithm transformation of the independent variables; and 3) with influential outliers excluded, a natural logarithm transformation of the independent variables and a logit transformation of the dependent variable. Review rate is statistically significant in all six cases for design reviews while controlling for developer ability and the initial work product quality.

The corresponding defect removal effectiveness models for *code reviews* are described in Table 10. Review rate is

TABLE 10
Code Review Effectiveness Models, Excluding Outliers, with Transformations

	C Data Set			C++ Data Set		
	Excluding Outliers	**Ln Transforms of X_i**	**Logit Transforms of Y**	**Excluding Outliers**	**Ln Transforms of X_i**	**Logit Transforms of Y**
Prob > F	<0.0001	<0.0001	<0.0001	<0.0001	<0.0001	<0.0001
R^2_a	0.0759	0.0873	0.0870	0.1610	0.2084	0.2092
	Coefficient estimates (standard errors)					
β_0	0.47**** (0.03)	0.62**** (0.09)	475.39**** (85.84)	0.48**** (0.03)	0.75**** (0.09)	618.35**** (92.46)
Ability	0.0007* (0.0004)	-0.03* (0.02)	-9.57 (18.07)	0.0009* (0.0005)	-0.02 (0.02)	-36.44 (23.37)
InitCodeQual	0.0009*** (0.0003)	-0.05** (0.02)	-115.90**** (19.97)	0.002**** (0.0003)	-0.11**** (0.02)	-151.79**** (21.87)
CRRate	0.03**** (0.005)	0.13**** (0.02)	62.84*** (21.08)	0.03**** (0.005)	0.18**** (0.02)	139.45**** (24.00)

*p<0.10; **p<0.05; ***p<0.01; ****p<0.001

again statistically significant in all six cases for code reviews while controlling for developer ability and the initial work product quality. Analogous to the relationship between Tables 4 and 6, White's test identified heteroscedasticity in the models in Table 7, but not in the transformed models in Tables 9 and 10. Finally, interaction effects were also investigated, but these more complex defect removal effectiveness models did not provide additional managerial insight [38].

In summary, both design and code review rates were found to be a statistically significant factor for the initial defect removal effectiveness models even after controlling for developer ability, the quality of the work product being reviewed, and the programming language used. Additional analyses with models that 1) excluded influential outliers, 2) used the logit transformation of the dependent variable for defect removal effectiveness, and 3) used a natural logarithm transformation of the independent variables provided a generally better fit, with review rate a statistically significant factor in all 12 cases for both design and code reviews and both languages. The managerial conclusions from these alternative models are consistent and not materially different from those for the initial simpler model form—review rate is strongly associated with defect removal effectiveness at statistically significant levels.

4.3 Mixed Models of Software Quality

In this section, we adopt an alternative approach to controlling for possible variance in developer (and reviewer) ability. Using multiple observations from the same developer could potentially affect the model. One simple way of addressing this concern would be to only use the data from a single assignment, but at a cost of discarding most of the data, and no additional insight is provided. A more sophisticated manner of dealing with multiple observations from the same developer is mixed models; their use is described below.

Rather than the basic measure of developer ability used above with the multiple regression models, we take advantage of the fact that with the PSP data set, there are multiple observations for each developer in the data, i.e., the final four assignments that include reviews. This issue is addressed in the context of repeated measures for a developer across assignments in mixed models, which allows us to incorporate more sophisticated per-developer regressions [15], [29], [31]. Mixed models can be used for analyzing growth curves using repeated data and they are an appropriate tool for analyzing models containing both fixed and random effects.[6]

The general linear mixed model equation [31] is

$$Y = X\beta + Zu + e, \tag{4}$$

where Y, X, β, and e are the same as the general linear model, and Z and u add in the random effects. Z is the random effects design matrix and u is the vector of random block effects. Random effects can also address learning curves via repeated measures for both the PSP course across the multiple assignments and individual students [31].

There are several reasons that a mixed model may be preferred over a fixed effects model [31], some of which include the following:

- The inference space for a statistical analysis can be characterized as narrow, intermediate, or broad, depending on how it deals with random effects. In the majority of practical applications, the broad inference space is of primary interest. The mixed model calculates broad inference space estimates and standard errors. The general linear model works with the narrow inference space. In the case of the PSP data, the broad inference space is appropriate since generalizing to the general population of the developers is desirable.
- Estimates in the general linear model are ordinary least squares. Estimates in the mixed model are estimated generalized least squares, which are theoretically superior.
- In the presence of random effects, the mixed model calculates correct standard errors by default, incorporating the variance components of random effects. The general linear model does not.
- Specific random effects, or linear functions of random effects, can be estimated using the best linear unbiased predictors (BLUPs), which are unique to mixed model theory. Fixed effect models use the best linear unbiased estimates (BLUEs). To the degree that random effects such as *years of experience* or *programming language* are significant in the mixed models, estimates incorporating all effects correctly, whether fixed or random, are appropriate.
- Observations with missing data for any repeated measures variable are discarded for the general linear model, where the mixed model can use all data present for a subject, so long as the missing data are random. Since any missing data for a subject causes all of the subject's data to be discarded in the general linear model, the power of the statistical tests is likely to be lower.

The mixed model deals with repeated measures, where multiple measurements of a response variable on the same subject are taken [29], [31]. For PSP, the repeated measures are taken on the students over the course of the class and the treatments (or between-subject) factors are the process changes that occur across PSP assignments. The objective of the analysis is to compare treatment means or treatment regression curves over time. Without considering repeated measures, the implicit assumption is that the covariances between observations on the same subject are the same, which may be unrealistic as observations close in time may be more highly correlated than those far apart in time.

The covariance structure for the repeated measures in a mixed model is specified by the analyst. A model-fit criterion, such as Akaike's Information Criterion (AIC),[7] is typically used to select the most appropriate covariance

6. Random effects describe data where only a subset of the levels of interest might have been included, for example, C and C++ are two of many programming languages. Potentially confounding variables, such as academic degrees and experience, can be explored as random effects.

7. AIC is essentially a log likelihood value penalized for the number of parameters estimated, therefore, it selects for simplicity and parsimony [29].

TABLE 11
Mixed Models for Defect Removal Effectiveness

	Design			Code	
	C Data Set	C++ Data Set		C Data Set	C++ Data Set
Prob > χ^2	0.0172	0.0042	**Prob > χ^2**	<0.0001	0.0518
AIC	350.6	229.2	***AIC***	39.4	4.7
Coefficient estimates (standard errors)					
β_0	0.60**** (0.05)	0.54**** (0.05)	**β_0**	0.43**** (0.03)	0.47**** (0.03)
Ability	0.001** (0.0006)	0.0002 (0.0009)	**Ability**	0.0005 (0.0004)	0.0005 (0.0006)
InitDsnQual	0.001 (0.0007)	0.002*** (0.0008)	**InitCodeQual**	0.0006** (0.0003)	0.001**** (0.0003)
DRRate	0.01** (0.006)	0.02*** (0.007)	**CRRate**	0.03**** (0.005)	0.03**** (0.005)

*$p<0.10$; **$p<0.05$; ***$p<0.01$; ****$p<0.001$

TABLE 12
Mixed Models for Design Review Effectiveness Excluding Outliers and with Transformations

	C Data Set			C++ Data Set		
	Excluding Outliers	Ln Transforms of X_i	Logit Transforms of Y	Excluding Outliers	Ln Transforms of X_i	Logit Transforms of Y
Prob > χ^2	0.0121	0.0054	0.0387	0.0039	0.0021	0.0345
AIC	347.2	306.1	5373.2	222.9	194.1	3472.7
Coefficient estimates (standard errors)						
β_0	0.60**** (0.05)	0.95**** (0.13)	1183.28**** (151.07)	0.57**** (0.06)	0.72**** (0.17)	745.86**** (184.93)
Ability	0.001** (0.0006)	-0.08** (0.03)	-69.67** (34.25)	0.0007 (0.001)	0.009 (0.05)	10.79 (49.21)
InitDsnQual	0.0009 (0.0007)	-0.10**** (0.03)	-251.37**** (31.59)	0.003**** (0.0008)	-0.12**** (0.03)	-192.90**** (33.62)
DRRate	0.01** (0.006)	0.15**** (0.03)	149.89**** (32.52)	0.02*** (0.007)	0.15**** (0.03)	94.15*** (35.49)

*$p<0.10$; **$p<0.05$; ***$p<0.01$; ****$p<0.001$

structure and model (smaller values are better), although a likelihood ratio test is generally considered to be more rigorous [29], [31]. For the mixed models described here, the unstructured covariance structure, which imposes no mathematical pattern on the covariance structure, is used. It has the smallest *AIC* value of all the covariance structures investigated[8] and the unstructured covariance structure was consistently better than the other structures for the likelihood ratio test. The model formulations in Table 11 mirror those of the regression models.

As was the case with the regression models, review rate is consistently and positively associated with defect removal effectiveness, and it is statistically significant at usual levels in all four cases while controlling for developer ability and the initial work product quality. Ability and work product quality are also positively associated with defect removal effectiveness, although they are not always statistically significantly different from zero.

The mixed models of defect removal effectiveness for design reviews are described in Table 12 corresponding to the design review models in Table 9. Review rate is statistically significant in all six cases for design reviews while controlling for developer ability and the initial work product quality. The model using a natural logarithm transformation of the dependent variables provides the best fit according to the AIC criterion.

The mixed models of defect removal effectiveness for code reviews are described in Table 13 corresponding to the regression models for code reviews in Table 10. Review rate is again statistically significant in all six cases for code reviews while controlling for developer ability and the initial code quality. The model using a natural logarithm

8. The covariance structures investigated were the unstructured, compound symmetry, autoregressive order 1, autoregressive order 1 with heterogeneous variances, compound symmetry with heterogeneous variances, spherical contrast, first-order factor analysis with specific variances all different, and first-order factor analysis with specific variances all the same [29].

TABLE 13
Mixed Models for Code Review Effectiveness Excluding Outliers and with Transformations

	C Data Set			C++ Data Set		
	Excluding Outliers	**Ln Transforms of X_i**	**Logit Transforms of Y**	**Excluding Outliers**	**Ln Transforms of X_i**	**Logit Transforms of Y**
Prob > χ^2	<0.0001	<0.0001	<0.0001	0.0518	0.0338	<0.0001
AIC	33.9	5.2	4852.1	-4.1	-36.1	3080.3
Coefficient estimates (standard errors)						
β_0	0.45^{****} (0.03)	0.58^{****} (0.09)	364.97^{****} (75.11)	0.49^{****} (0.03)	0.81^{****} (0.11)	493.64^{****} (87.10)
Ability	0.0007 (0.0004)	-0.02 (0.02)	-12.11 (16.48)	0.0008 (0.0006)	-0.02 (0.03)	-31.88 (21.76)
InitCodeQual	0.0007^{**} (0.0003)	-0.05^{**} (0.02)	-88.42^{****} (16.71)	0.002^{****} (0.0004)	0.12^{****} (0.02)	-116.17^{****} (18.51)
CRRate	0.03^{****} (0.005)	0.16^{****} (0.02)	57.02^{***} (17.75)	0.03^{****} (0.005)	0.18^{****} (0.02)	93.39^{****} (19.21)

$^{*}p<0.10$; $^{**}p<0.05$; $^{***}p<0.01$; $^{****}p<0.001$

transformation of the dependent variables again provides the best fit according to the AIC criterion.

The managerial conclusions suggested by the traditional regression models and also supported by the more rigorous mixed models as design and code review rates are positive and statistically significant in all 16 cases. However, the mixed models do address the issue of independence of the observations when there are multiple observations for a single student, and they address the systemic changes in performance across assignments via repeated measures. This statistical rigor allows greater confidence in the results.

4.4 Sensitivity Analysis

Mixed models also support investigation of random effects that might confound the analyses. Six potentially confounding factors were investigated. A separate set of random effect models was built for each random effect investigated, using the mixed models previously described for the fixed effects. The 243 distinct developers reported up to 34 years of experience, with a median of seven years of experience. They were typically familiar with four programming languages and some were familiar with over 10. Almost all of the developers were college graduates; there were 138 bachelors' degrees, 71 masters' degrees, and nine doctorates in the group.

None of the following five variables was found to be statistically significant as follows:

- *highest academic degree attained* (of BA, BS, MS, PhD);
- *years of experience;*
- *number of programming languages known;*
- *programming language used;* and
- *number of assignments completed by each developer* (some developers did not finish all 10 assignments).

However, the *percentage of time spent programming* was found to be positive and statistically significant ($p = 0.0358$) as a random effect in one of the four tests (code reviews in C), although it was not significant in the other three cases: design reviews in C, design reviewers in C++, and code reviews in C++. However, even in the code review in C case, including this variable in the model did not materially affect the significance levels of the variables shown to explain variance in the earlier models, and therefore, does not change any of the managerial conclusions already drawn. With a total of 24 tests (6 variables times 4 models each), a single test represents 4 percent (0.04) of the data. At the 0.05 level of statistical significance, a single "false positive" could be expected by chance about once in 20 times, and therefore, there can be little confidence that this one variable in one test is a meaningful result. Overall, since these variables do not contribute to the models, they are not included in the analyses that follow.

4.5 Testing the Recommended Review Rate

Inspections include both *preparation* rates (individual inspector) and *inspection* rates (meeting time for the inspection team). Review rates in PSP correspond to preparation rates in inspections since preparation is performed by individuals rather than teams. The recommended preparation rate is less than or equal to 200 LOC/hour [27], [47]. A faster rate is considered ineffective and a reinspection is recommended. This provides two classes of reviews based on review rate: those where the review rate is greater than the recommended limit of 200 LOC/hour ("fast review rate") and reviews less than or equal to the recommended rate ("recommended review rate").

The estimates of the means for defect removal effectiveness at the recommended review rate versus a faster rate are listed in Table 14 for both design and code reviews and for both the C and C++ data sets. The Levene test revealed an undesirable level of heteroscedasticity for the four models in Table 14 due to "boundary peaks" at 0 and 100 percent, so a Welch test was used to test for statistical significance in the presence of heteroscedasticity. In all four cases, reviewers who adhere to the recommended review rate find more defects than those who do not. For design defects, they find 66 and 57 percent, respectively,

TABLE 14
Defect Removal Effectiveness for Recommended versus Fast Review Rates

Review Rate Category	Design Review Rate		Code Review Rate	
	C Data Set	C++ Data Set	C Data Set	C++ Data Set
Recommended Review Rate	65.6 (3.6)	56.7 (4.7)	56.0 (2.3)	57.4 (2.8)
Fast Review Rate	48.8 (2.3)	50.7 (2.8)	46.9 (1.6)	45.2 (1.8)
p-value (Welch)	<0.0001	0.2206	0.0013	0.0003
power (α=0.05)	0.9770	0.1944	0.9031	0.9545

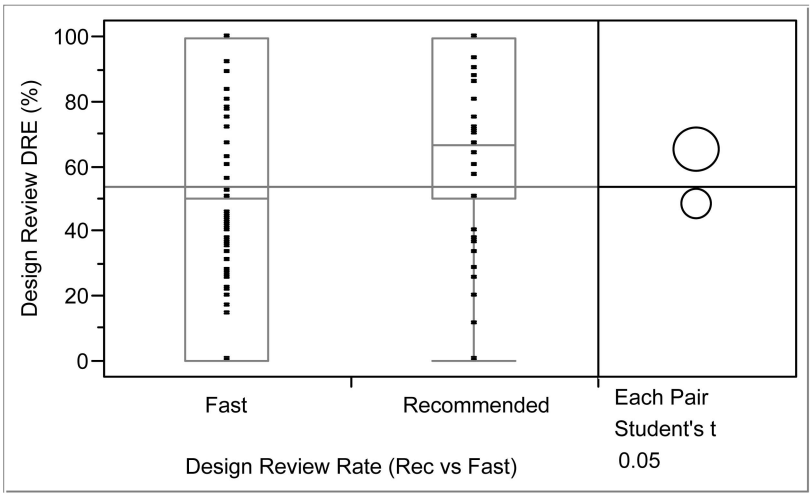

Fig. 5. Comparing recommended versus fast design review rates for C.

and 56 and 57 percent for code defects. Reviewers using a faster-than-recommended review rate find only 49 and 51 percent of design defects and 47 and 45 percent of code defects. This superior performance difference is statistically significant at a very high level of statistical significance in three of the four cases, and in the one case that is not statistically significant, the power of the statistical test is low. Whether these differences are economically significant will depend on the business situation; in the PSP context, this difference may be as few as one or two defects since the absolute numbers are not large.

In order to provide some greater intuition, these results are illustrated graphically in Fig. 5 for the design review rates using the C data set. It is interesting to note that this statistically significant difference is present despite the fact that reviews at both the recommended and fast rates reflect variances in performance that range from 0 to 100 percent in their effectiveness.

Overall, these results tend to support the recommended preparation rates, but they also raise the question as to whether a slower rate would be even more effective. In order to test this additional question, the data on review rates were categorized into 100-LOC and 50-LOC "bins." For reviews using the recommended review rates, no statistically significant difference was observed for either design or code reviews for review rates at 0-100 LOC/hour versus 100-200 LOC/hour; neither was a statistically significant difference observed between 50-LOC bins. As shown in Fig. 6, however, defect removal effectiveness continues to decline as the review rate becomes faster.

4.6 Threats to External Validity

Concerns are sometimes raised as to whether data from assignments done by students can be generalized to industry projects. However, for PSP, two factors tend to alleviate this concern. First, PSP classes are frequently taught in an industry rather than an academic setting, and the developers in our sample reported up to 34 years of experience, with a median of seven years of experience.[9] Therefore, these "students" look much more like average industry developers rather than the canonical computer science undergraduate subjects.

Second, since the target of this analysis is preparation rate for inspections, and the recommended rate is about 100 LOC/hour (maximum of 200 LOC/hour), and the recommended length of an inspection meeting that the developer would prepare for is 2 hours, the typical size of work product we might expect to see in an industry inspection would be about 200 LOC. Since the median size of a PSP assignment is about 120 LOC, the relative sizes of the work products being reviewed are comparable, although smaller. Therefore, although industrial projects on the whole are, of course, much larger than the typical PSP assignment, the task size that is most relevant here is about the same.

9. Note that, as reported in Section 4.4, amount of experience was not a significant factor affecting the quality of PSP work.

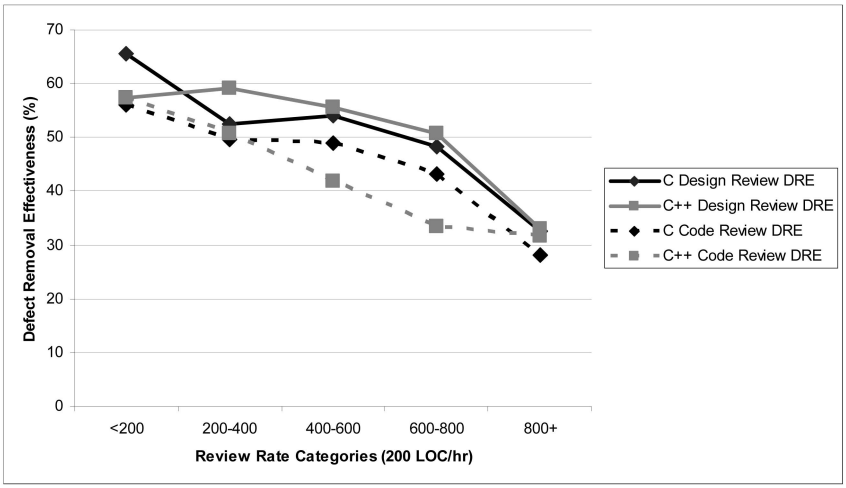

Fig. 6. Defect removal effectiveness versus review rate.

Of course, as a general rule, any process or tool should be calibrated to its environment and it would be advisable for any organization to empirically determine the preferred review rates for its developers in its own environment. It is possible that the relative smaller size of PSP assignments may be an influence on these results. In addition, in this research, the only reviewer is the developer, whereas in larger commercial applications, the developer is typically just one of a team of inspectors. The lack of nondeveloper reviewers may be an influence on the results. However, the recommended rates provide a useful starting point for this investigation and our research independently supports earlier recommendations.

5 Discussion, Implications, and Conclusions

This research supports and quantifies the notion that the quality of software depends upon the quality of the work done throughout the life cycle. While the definition of quality in a larger context includes many attributes other than defects, understanding defects helps us to control and improve our processes. This research shows that the quality of the work products depends upon the quality control mechanisms we employ as an integral part of our processes, and it specifically supports the view that review rate affects the defect removal effectiveness of reviews. Since PSP review rates are analogous to the preparation rates in inspections, it provides empirical evidence supporting the notion that allowing sufficient preparation time is a significant factor for more effective inspections while controlling for other factors that might be expected to impact performance. The data also support the observation that review quality declines when the review rate exceeds the recommended maximum of 200 LOC/hour. And, it is also worth noting that the performance of PSP's checklist-based individual reviews is similar to the 60 percent defect removal effectiveness reported for inspections performed by recently trained teams [47].

It may be somewhat surprising to some that, once the 200 LOC/hour limit has been met, a more deliberate pace seems to not materially improve performance. Of course, whether an improvement in effectiveness of 8-15 percent for design reviews and 10-11 percent for code reviews is economically worthwhile is a business decision (similarly for the 25-33 percent difference for review rates greater than 800 LOC/hour); this research quantifies the difference in order to permit informed decisions to be made in terms of process changes. However, given the increasing reliance of organizations and products on high-quality software, and the typically significant negative consequences of defective software, it is to be expected that such a level of improvement is likely to be cost-effective for the vast majority of organizations.

Statistical analyses can identify a correlation, but they do not prove a cause-and-effect relationship. An alternative explanation to the idea that slowing the review rate is the reason that more issues are found might be that when a developer finds more issues, he or she slows down to address the defect (e.g., recording it).

However, when Fagan formalized the inspection process in 1976, he included guidelines on recommended preparation and meeting rates for effective inspections based on his observations at IBM [17]. Buck found an optimal inspection rate of 125 LOC/hour, and rates faster than this showed a rapid decline in the number of defects found [9]. Weller found similar results for preparation rate [50]. Christenson and Huang modeled the inspection process based on the premise that with more effort, a greater fraction of coding errors will be found [10]. After considering the findings of these and other researchers, Radice concluded that "it is almost one of the laws of nature about inspections, i.e., the faster an inspection, the fewer defects removed" [47]. Although many factors may affect the effectiveness of a review, both conceptually and empirically, it is clear that review rate is an important factor, and we follow the precedent of previous research in attributing the higher number of defects found to the care taken in performing the review.

The statistical insight provided by this research is possible because of the rich data set available in PSP for empirically investigating important process issues. PSP provides an opportunity to retrospectively analyze design and code reviews, while allowing for the control of a variety of factors, including developer ability and programming language,

which might be expected to influence the results. The size and longitudinal nature of the PSP data sets allow us to use statistical techniques, such as mixed models, that enable us to address issues such as multiple observations for a developer. Mixed models also allow us to rigorously account for potentially confounding factors, such as years of experience.

When effects are relatively small and the amount of variation in the process is intrinsically large, it can be difficult to discern their significance. One of the advantages of the PSP data is that it provides a thoroughly instrumented set of observations for large data sets. Few industry projects provide similar amounts of data with the degree of process instrumentation built into PSP. Extending this research into industry projects is a logical expectation. One possible source of rich process data from industry is the Team Software Process (TSP), which applies and extends the PSP concepts to project teams [26]. As TSP is adopted by industry, we can hope to perform similar sophisticated analyses across multiple TSP projects. This would allow us to investigate a larger subset of the factors that affects inspection effectiveness. Instructors of the PSP course might also consider teaching other forms of inspection, e.g., using scenario-based rather than checklist-based inspections.

This research empirically verifies that allowing sufficient preparation time for reviews and inspections can produce better performance. Within a deadline-driven business context, it is easy to see examples of the tendency to take shortcuts, which, while perhaps meeting a short-term schedule goal, have clear implications for raising future costs of maintenance and customer support for the resulting lower quality software. We can only hope that empirical evidence supporting recommended practices will drive economically optimal behavior within projects. The message is clear: Disciplined processes that follow recommended practice can improve performance.

REFERENCES

[1] A.F. Ackerman, L.S. Buchwald, and F.H. Lewski, "Software Inspections: An Effective Verification Process," *IEEE Software,* vol. 6, no. 3, pp. 31-36, May/June 1989.

[2] F. Akiyama, "An Example of Software System Debugging," *Proc. Int'l Federation for Information Processing Congress 71,* pp. 353-359, Aug. 1971.

[3] R.D. Banker, S.M. Datar, and C.F. Kemerer, "A Model to Evaluate Variables Impacting Productivity on Software Maintenance Projects," *Management Science,* vol. 37, no. 1, pp. 1-18, Jan. 1991.

[4] D.A. Belsley, E. Kuh, and R.E. Welsch, *Regression Diagnostics: Identifying Influential Data and Sources of Collinearity.* John Wiley & Sons, 1980.

[5] B.W. Boehm, C. Abts, A.W. Brown, S. Chulani, B.K. Clark, E. Horowitz, R.J. Madachy, D. Reifer, and B. Steece, *Software Cost Estimation with COCOMO II.* Prentice-Hall, 2000.

[6] K.V. Bourgeois, "Process Insights from a Large-Scale Software Inspections Data Analysis," *Crosstalk: J. Defense Software Eng.,* vol. 9, no. 10, pp. 17-23, Oct. 1996.

[7] F.W. Breyfogle III, *Implementing Six Sigma: Smarter Solutions Using Statistical Methods.* John Wiley & Sons, 1999.

[8] S. Brocklehurst and B. Littlewood, "Techniques for Prediction Analysis and Recalibration," *Handbook of Software Reliability Engineering,* M.R. Lyu, ed., pp. 119-166, IEEE CS Press, 1996.

[9] F.O. Buck, "Indicators of Quality Inspections," IBM Technical Report TR21.802, Systems Comm. Division, Dec. 1981.

[10] D.A. Christenson and S.T. Huang, "A Code Inspection Model for Software Quality Management and Prediction," *Proc. IEEE Global Telecomm. Conf. and Exhibition in Hollywood,* pp. 14.7.1-14.7.5, Nov. 1988.

[11] M. Criscione, J. Ferree, and D. Porter, "Predicting Software Errors and Defects," *Proc. 2001 Applications of Software Measurement,* pp. 269-280. Feb. 2001,

[12] B. Curtis, "The Impact of Individual Differences in Programmers," *Working with Computers: Theory versus Outcome,* G.C. van der Veer, ed., pp. 279-294, 1988.

[13] B. Curtis, H. Krasner, and N. Iscoe, "A Field Study of the Software Design Process for Large Systems," *Comm. ACM,* vol. 31, no. 11, pp. 1268-1287, Nov. 1988.

[14] W.E. Deming, *Out of the Crisis.* MIT Center for Advanced Eng. Study, 1986.

[15] T.E. Duncan, S.C. Duncan, L.A. Strycker, F. Li, and A. Alpert, *An Introduction to Latent Variable Growth Curve Modeling.* Lawrence Erlbaum Assoc., 1999.

[16] S.G. Eick, C.R. Loader, M.D. Long, S.A.V. Wiel, and L.G. Votta, "Estimating Software Fault Content before Coding," *Proc. 14th Int'l Conf. Software Eng.,* May 1992.

[17] M.E. Fagan, "Design and Code Inspections to Reduce Errors in Program Development," *IBM Systems J.,* vol. 15, no. 3, pp. 182-211, 1976.

[18] M.E. Fagan, "Advances in Software Inspections," *IEEE Trans. Software Eng.,* vol. 12, no. 7, pp. 744-751, July 1986.

[19] N. Fenton and M. Neil, "A Critique of Software Defect Prediction Models," *IEEE Trans. Software Eng.,* vol. 25, no. 5, pp. 675-689, Sept./Oct. 1999.

[20] P. Ferguson, W.S. Humphrey, S. Khajenoori, S. Macke, and A. Matvya, "Results of Applying the Personal Software Process," *Computer,* vol. 30, no. 5, pp. 24-31, May 1997.

[21] T. Gilb and D. Graham, *Software Inspection.* Addison-Wesley, 1993.

[22] G.K. Gill and C.F. Kemerer, "Cyclomatic Complexity Density and Software Maintenance Productivity," *IEEE Trans. Software Eng.,* vol. 17, no. 12, pp. 1284-1288, Dec. 1991.

[23] R.L. Glass, "Inspections—Some Surprising Findings," *Comm. ACM,* vol. 42, no. 4, pp. 17-19, Apr. 1999.

[24] W. Hayes and J.W. Over, "The Personal Software Process (PSP): An Empirical Study of the Impact of PSP on Individual Engineers," Technical Report CMU/SEI-97-TR-001, Software Eng. Inst., Carnegie Mellon Univ., 1997.

[25] W.S. Humphrey, *A Discipline for Software Engineering.* Addison-Wesley, 1995.

[26] W.S. Humphrey, *Introduction to the Team Software Process.* Addison-Wesley, 1999.

[27] *IEEE 1028, IEEE Standard for Software Reviews and Audits,* IEEE CS, Aug. 2008.

[28] P.M. Johnson and A.M. Disney, "The Personal Software Process: A Cautionary Case Study," *IEEE Software,* vol. 15, no. 6, pp. 85-88, Nov./Dec. 1998.

[29] R. Khattree and D.N. Naik, *Applied Multivariate Statistics with SAS Software.* SAS Publishing, 1999.

[30] L.P.K. Land, "Software Group Reviews and the Impact of Procedural Roles on Defect Detection Performance," PhD dissertation, Univ. of New South Wales, 2002.

[31] R.C. Littell, G.A. Milliken, W.W. Stroup, and R.D. Wolfinger, *SAS System for Mixed Models.* SAS Publishing, 1996.

[32] *Handbook of Software Reliability Engineering,* M.R. Lyu, ed. IEEE CS Press, 1996.

[33] R.T. McCann, "How Much Code Inspection Is Enough?" *Crosstalk: J. Defense Software Eng.,* vol. 14, no. 7, pp. 9-12, July 2001.

[34] J. Neter, M.H. Kutner, and C.J. Nachtscheim, *Applied Linear Statistical Models,* fourth ed. Irwin, 1996.

[35] D.L. Parnas and D.M. Weiss, "Active Design Reviews: Principles and Practices," *J. Systems Software,* vol. 7, no. 4, pp. 259-265, Dec. 1987.

[36] M.C. Paulk, C.V. Weber, B. Curtis, and M.B. Chrissis, *The Capability Maturity Model: Guidelines for Improving the Software Process.* Addison-Wesley, 1995.

[37] M.C. Paulk, "The Evolution of the SEI's Capability Maturity Model for Software," *Software Process: Improvement Practice,* vol. 1, no. 1, pp. 3-15, Spring, 1995.

[38] M.C. Paulk, "An Empirical Study of Process Discipline and Software Quality," PhD dissertation, Univ. of Pittsburgh, 2005.

[39] J.M. Perpich, D.E. Perry, A.A. Porter, L.G. Votta, and M.W. Wade, "Anywhere, Anytime Code Inspections: Using the Web to Remove Inspection Bottlenecks in Large-Scale Software Development," *Proc. 19th Int'l Conf. Software Eng.,* pp. 14-21, May 1997.

[40] D.E. Perry, A.A. Porter, M.W. Wade, L.G. Votta, and J. Perpich, "Reducing Inspection Interval in Large-Scale Software Development," *IEEE Trans. Software Eng.*, vol. 28, no. 7, pp. 695-705, July 2002.
[41] A.A. Porter, H.P. Siy, and L.G. Votta, "Understanding the Effects of Developer Activities on Inspection Interval," *Proc. 19th Int'l Conf. Software Eng.*, pp. 128-138. May 1997.
[42] A.A. Porter, H.P. Siy, C.A. Toman, and L.G. Votta, "An Experiment to Assess the Cost-Benefits of Code Inspections in Large Scale Software Development," *IEEE Trans. Software Eng.*, vol. 23, no. 6, pp. 329-346, June 1997.
[43] A.A. Porter and P.M. Johnson, "Assessing Software Review Meetings: Results of a Comparative Analysis of Two Experimental Studies," *IEEE Trans. Software Eng.*, vol. 23, no. 3, pp. 129-145, Mar. 1997.
[44] A.A. Porter and L.G. Votta, "What Makes Inspections Work?" *IEEE Software,* vol. 14, no. 6, pp. 99-102, Nov./Dec. 1997.
[45] A.A. Porter, H.P. Siy, A. Mockus, and L.G. Votta, "Understanding the Sources of Variation in Software Inspections," *ACM Trans. Software Eng. and Methodology,* vol. 7, no. 1, pp. 41-79, Jan. 1998.
[46] L. Prechelt and B. Unger, "An Experiment Measuring the Effects of Personal Software Process (PSP) Training," *IEEE Trans. Software Eng.,* vol. 27, no. 5, pp. 465-472, May 2000.
[47] R.A. Radice, *High Quality Low Cost Software Inspections.* Paradoxicon Publishing, 2002.
[48] J.O. Rawlings, S.G. Pantula, and D.A. Dickey, *Applied Regression Analysis,* second ed. Springer-Verlag, 1998.
[49] M. Takahasi and Y. Kamayachi, "An Empirical Study of a Model for Program Error Prediction," *Proc. Eighth Int'l Conf. Software Eng.,* pp. 330-336. Aug. 1985.
[50] E.F. Weller, "Lessons from Three Years of Inspection Data," *IEEE Software,* vol. 10, no. 5, pp. 38-45, Sept. 1993.
[51] A. Wesslen, "A Replicated Empirical Study of the Impact of the Methods in the PSP on Individual Engineers," *Empirical Software Eng.,* vol. 5, no. 2, pp. 93-123, June 2000.
[52] C. Withrow, "Error Density and Size in Ada Software," *IEEE Software,* vol. 7, no. 1, pp. 26-30, Jan. 1990.
[53] C. Wohlin and P. Runeson, "Defect Content Estimations from Review Data," *Proc. 20th Int'l Conf. Software Eng.,* pp. 400-409, Apr. 1998.
[54] C. Wohlin and A. Wesslen, "Understanding Software Defect Detection in the Personal Software Process," *Proc. Ninth Int'l Symp. Software Reliability,* pp. 49-58, Nov. 1998.
[55] C. Wohlin, "Are Individual Differences in Software Development Performance Possible to Capture Using a Quantitative Survey?" *Empirical Software Eng.*, vol. 9, no. 3, pp. 211-228, Sept. 2004.

Chris F. Kemerer received the BS degree from the Wharton School at the University of Pennsylvania and the PhD degree from Carnegie Mellon University. He is the David M. Roderick Chair in Information Systems at the Katz Graduate School of Business at the University of Pittsburgh. Previously, he was an associate professor at the Massachusetts Institute of Technology's Sloan School of Management. His current research interests include management and measurement issues in information systems and software engineering, and the adoption and diffusion of information technologies. He has published more than 60 papers on these topics, as well as editing two books. He has served on a number of editorial boards, including serving as the departmental editor for *Management Science* and as the editor-in-chief of *Information Systems Research*. He is a member of the IEEE Computer Society and is a past associate editor of the *IEEE Transactions on Software Engineering*.

Mark C. Paulk received the MS degree in computer science from Vanderbilt University and the PhD degree in industrial engineering from the University of Pittsburgh. He is a senior systems scientist in the Institute for Software Research at Carnegie Mellon University. He led the team at the Software Engineering Institute that developed the Capability Maturity Model for Software and was the coproject editor for ISO/IEC 15504:2 (Performing an Assessment). His research interests are high maturity practices, statistical thinking for software processes, empirical research into best practices, and agile methods. He is a senior member of the IEEE.