AN EMPIRICAL STUDY OF PROCESS DISCIPLINE AND SOFTWARE QUALITY

by

Mark Christopher Paulk

BS, University of Alabama in Huntsville, 1978

MS, Vanderbilt University, 1980

Submitted to the Graduate Faculty of

the School of Engineering in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

University of Pittsburgh

2005

UNIVERSITY OF PITTSBURGH

SCHOOL OF ENGINEERING

This dissertation was presented

by

Mark Christopher Paulk

It was defended on

July 13, 2004

and approved by

Dr. Kim LaScola Needy, Associate Professor, Department of Industrial Engineering

Dr. Jayant Rajgopal, Associate Professor, Department of Industrial Engineering

Dr. Chris Kemerer, Professor, Joseph M. Katz Graduate School of Business

Dr. Marc Kellner, Senior Member of the Technical Staff, Software Engineering Institute

Dissertation Director: Dr. Mainak Mazumdar, Professor, Department of Industrial Engineering

AN EMPIRICAL STUDY OF PROCESS DISCIPLINE AND SOFTWARE QUALITY

Mark Christopher Paulk, PhD

University of Pittsburgh, 2005

There is a widespread, but not universal, belief in the software community that software organizations and projects can systematically improve their ability to meet commitments and build high-quality products using principles of software quality management. Quality affects cost and schedule, therefore the engineering practices that affect quality are also a management concern. Understanding the factors that influence software quality is crucial to the continuing maturation of the software industry; an improved understanding of software quality drivers will help software engineers and managers make more informed decisions in controlling and improving the software process.

My research is motivated by a desire to understand the effect of disciplined processes and effective teams on improving performance and lessening variability with respect to software quality. Classroom data provides insight into interpersonal differences between competent professionals as increasingly disciplined processes are adopted. Project data using similar processes enables an exploration of the impact of effective teams on software quality.

My results show that:

- Program size, programmer ability, and disciplined processes significantly affect software quality.
- Factors frequently used as surrogates for programmer ability, e.g., *years of experience,* and technology, e.g., *programming language*, do not significantly impact software quality.

- Recommended practices are not necessarily followed even when processes are consistently performed, e.g., peer reviews may be consistently performed, but the review rates may exceed recommended practice for effective reviews.
- When moving from *ad hoc* processes to disciplined processes, top-quartile performers improve more than 2X; bottom-quartile performers improve more than 4X.
- Rigorous statistical techniques that allow for individual differences confirm the importance of process discipline and following recommended practice for improving software quality.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

To my wife Kathy
Without your encouragement and support, this would never have happened.

PREFACE

I am grateful to many people who have supported me throughout my doctoral study. First, I would like to thank Dr. Mainak Mazumdar for reviewing my work and guiding the course of my research. His focus was always on the quality of the research.

I am indebted to the members of my dissertation committee: Dr. Kim Needy, Dr. Jayant Rajgopal, Dr. Chris Kemerer, and Dr. Marc Kellner. Their feedback improved this dissertation greatly. I am also grateful to Dr. John Manley, my initial advisor, who sent me down this road.

I am thankful to Bill Peterson, Watts Humphrey, and Jim Over of the Software Engineering Institute, who provided access to the Personal Software Process data, as well as all of the PSP students. The data was crucial to my research, and I hope my results support their ongoing work in transforming the software industry.

Many thanks to the organizations and projects that provided me with data. The substance of research is ultimately realized in how industry incorporates the results into the way business is done. The heart of software process improvement is in its transformation of how software is built and the resulting improvements in productivity and quality.

Last, but not least, I am indebted to my wife, Kathy, and my son, David, for their patience and encouragement. Their support provided my foundation.

1.0 INTRODUCTION

1.1 MOTIVATION FOR THIS RESEARCH

During the last four decades, there have been recurring complaints by customers and executives about software projects that are chronically late and over-budget and about software products that are of unsatisfactory quality – to the extent that customers speak of the "software crisis" [Gibbs 1994]. A recent study by the National Institute of Standards and Technology suggests that costs associated with software quality problems annually range between $22.2 and $59.5 billion [National Institute of Standards and Technology 2002]. During the last two decades, there have been systematic attempts to address the software crisis by applying the concepts of Total Quality Management (TQM) and industrial engineering to software projects.

Quality affects cost and schedule; the engineering practices that affect quality are therefore a management concern. The number of defects in a software product is a quantitative measure for software quality. Although "quality" includes other attributes, such as availability, features, and cost, 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.

The broad-scale use of the concepts of TQM, including rigorous statistics and statistical process control (SPC), in design-intensive, human-centric processes such as those for building software is a relatively recent phenomenon. There is now a widespread, but not universal, belief

in the software community that software organizations and projects can systematically improve their ability to meet commitments and build high-quality products by following fundamental principles of software quality:

- *Competent Professionals*. Competent professionals, who are trained and experienced in software engineering methodologies and relevant application domains, are essential for high-quality work [DeMarco and Lister 2003].
- *Project Management*. Effective project management enables the consistent performance of effective engineering practices [Humphrey 1989].
- *Techniques and Tools*. Software professionals need effective and appropriate techniques and tools to do high-quality work [Humphrey 1989].
- *Disciplined Processes*. Disciplined processes, which are consistently performed by competent professionals, lead to high-quality work [Paulk et al. 1995; Sawyer and Guinan 1998].
- *Quantitative Management*. Quantitative management of disciplined processes, which includes the use of rigorous statistical techniques, enables informed decision making by both managers and engineers, i.e., management by fact [Florac and Carleton 1999].
- *Effective Teams*. Effective teams, which are so strongly knit together that the whole is greater than the sum of the parts, do superior work with less variation than individuals or "ad hoc" teams [DeMarco and Lister 1999, 123].

Most software professionals would agree with the tenor of these principles, although there are many barriers to effectively implementing them. Organizational politics, cynicism by the staff, resistance to change, and dysfunctional customer-supplier relationships are among the challenges software organizations must deal with in building high-quality products [Goldenson

and Herbsleb 1995; Beer, Eisenstat, and Spector 1990; Besselman, Arora, and Larkey 1995]. Software organizations are only beginning to apply quality management concepts that have been well-known in manufacturing for decades, and the preferred quantitative techniques remain sharply debated.

Three schools of thought can be observed in the software community with respect to statistics and quantitative management. Adherents of the first school consider rigorous statistical analysis inappropriate for software processes and products. They consider the software process to be a design activity, an intellectual and social activity subject to many influences, that manufacturing-style statistical process control (SPC) cannot be applied to [Ould 1996]. They recommend simple graphical tools and engineering judgment.

The second school includes some of the most respected names in software engineering. After years of attempting to characterize software projects using classical statistical techniques, such as regression analysis, they have concluded the classical techniques are inappropriate because of the scarcity of data, the large variation in individual performance, and a lack of consistently applied operational definitions. They recommend alternative statistical techniques such as non-parametric analyses [Curtis 1981], Bayesian Belief Networks that incorporate expert judgment [Fenton and Neil 1999], and a "relaxed" use of statistics, e.g., pseudo-control charts [Kan 2003, 145].

The third school, which I belong to, acknowledges the challenges identified by the members of the second school but argues that classical statistical techniques can be useful if certain prerequisites are satisfied: software process data for statistical analysis must come from competent professionals using disciplined and conformant processes in effective teams. A ***disciplined*** process can be defined as a set of activities that is consistently performed to achieve

some given purpose. If the disciplined process is consistently performed in accordance with a set of identified requirements, it can also be characterized as a ***conformant*** process. (A ***mature*** process is both measured and measurably improving; the term is typically used for organizational standard processes since the baseline against which improvement is measured is typically established by organizations engaged in process improvement.) Rigorous studies of the effective use of statistical techniques for controlling the software process are needed to help overcome the resistance to applying sophisticated quality management principles to the software process.

1.2 STATEMENT OF THE SOFTWARE QUALITY PROBLEM

In the last two decades, the focus of software process improvement has been on addressing the principle of disciplined processes by implementing fundamental project management and organizational learning practices. Unrealistic plans and over-commitments lead to abandoning good engineering practice in the resultant schedule crunch, which in turn leads to inconsistent execution and poor software quality. A number of "best practices" are known in software engineering, but self-discipline is difficult, and imposing discipline on software professionals externally without their buy-in and commitment is impractical.

For example, the most powerful defect identification technique known for software engineering is a review of a work product by the peers of its producer called an inspection [Fagan 1976; Fagan 1986]. Recommended preparation rates, inspection rates, team size, etc., specify the preferred inspection process, although there are variants [Glass 1999]. Undisciplined processes are intrinsically unstable; for example, inspections that do not follow the pertinent rules are inconsistent and ineffective. Peer reviews that are consistently performed can be considered disciplined, but they must satisfy the specified inspection rules to be conformant to

the inspection process. The defect removal effectiveness (the percentage of defects ultimately found) of testing and walkthroughs is on the order of 30% [Jones 1996]; the defect removal effectiveness of inspections typically ranges from 60-90% [Fagan 1986, 750]. If the defect removal effectiveness of an inspection process improves over time for an organization, the measurable improvement is an objective indicator that the inspection process is mature.

In recent years, a growing number of software organizations has begun to focus on quantitative management, which implies an understanding of variation. The state-of-the-practice in software engineering is not yet sufficiently advanced to assume that rigorous statistical techniques are being consistently and correctly applied, even when a process is reputed to be "quantitatively managed" [Paulk and Chrissis 2000]. Disciplined and conformant processes are generally considered a prerequisite for quantitative management, along with competent professionals performing the work and effective teamwork.

Competent professionals are necessary for high-quality work, but the range of performance between individuals can span an order of magnitude [DeMarco and Lister 1999, 45]. Frequently a manager or supervisor has little control over who is assigned to his or her team. Effective teams perform better and with less variation than individuals [Hare et al. 1995], therefore appropriately-formed teams and disciplined processes are considered prerequisites for quantitative management. Effective teams are considered a prerequisite for quantitative management to occur in any rigorous statistical sense, yet the optimum size of an inspection team, for example, remains a controversial topic in spite of over 25 years of research [Radice 2002, 314].

Without competent professionals and disciplined processes, variability is so great that quantitative management provides little useful insight. Without appropriate techniques and tools or in the absence of effective teams, data simply highlights the ineffectiveness of the process.

It is specifically the use of "rigorous statistics" that arouses controversy, however. Rigorous is difficult to define precisely, but in the context of my research it implies an explicit understanding of variation. Averages and trend lines are quantitative; averages with confidence intervals and trend lines with prediction intervals are rigorous. Intervals, control limits, and other techniques for bounding variability sharpen the decision making process by setting expectations for what is usual and what is atypical.

Design-intensive work inherently has high variability; processes are potentially repeatable, even if they are not repetitive in the assembly line sense. Although the application of rigorous statistical techniques to software development is skeptically viewed by some, a number of organizations has demonstrated that statistical process control (SPC) can be applied to software processes [Florac and Carleton 1999; Paulk, Goldenson, and White 2000; Paulk and Chrissis 2000; Florac, Carleton, and Barnard 2000; Weller 2000]. The software profession cannot be considered an engineering discipline without a firm foundation in the use of quantitative data [Shaw 1990, 15].

In addition to the high variability intrinsic to software processes, many of the statistical tools that support decision-making incorporate assumptions about the statistical distributions followed by the data. For example, u-charts assume that (software) defect data follow a Poisson distribution. Relatively few papers have been published in the software field with experimentally validated results [Tichy et al. 1995], and assumptions have been used that are plausible but not empirically validated. Inconsistent, contradictory, and counter-intuitive

research results compounded by poor theory and methodology are widespread problems in software experimentation [Fenton, Pfleeger, and Glass 1994]. Published results are inadequate for supporting or denying the various statistical assumptions that have been made, especially when exacerbated by high variability. As a result, software organizations striving to apply rigorous statistical techniques have inconsistent and contradictory advice on which statistical assumptions it is reasonably safe to make.

Statistical control surfaces a number of issues that impact the quality of the product, such as complexity of the application domain, competence and experience of the people doing the work, power of the tools and support environment, etc. Covariates that affect product quality may confound a statistical analysis unless those variables are appropriately factored in.

My research is therefore motivated by a desire to understand the effect of using disciplined processes and effective teams on lessening the intrinsic variability of individual performance in the software process, specifically with respect to the statistical characteristics of software defects. While I agree on the importance of competent professionals for software quality, separating the effect of individual differences from that of disciplined processes should refute those who argue against a process focus [Bach 1994] and demonstrate the feasibility of using rigorous and sophisticated statistical techniques on software process data.

The use of classroom data, obtained when teaching about disciplined personal processes, provides insight into interpersonal differences between competent professionals when using appropriate techniques and tools and disciplined processes. Demographic data for the students permits an exploration of the factors affecting individual performance, and increasingly sophisticated processes permit an exploration of the effect of disciplined processes. Data from

industry projects allows analysis of mature software processes, thereby exploring the impact of inspection teams.

1.3 PURPOSE AND SIGNIFICANCE OF THIS STUDY

The purpose of this study is to verify and quantify the common wisdom that process discipline and effective teams improve performance and decrease variability for software quality. The definition of quality is limited to "conformance to requirements," thereby excluding issues associated with requirements elicitation and volatility.

Although the specific parameters that characterize organizational environments or application domains may differ, when reasonable statistical characterizations of software processes are determined for disciplined processes, engineers will be better able to identify appropriate statistical control and process modeling techniques to support making day-to-day decisions. In the presence of skewed distributions, the preferred statistical technique is likely to be one that takes appropriate advantage of the underlying distribution of the data [Porter 2001; Das 2003; Mullen 1998].

My research uses relatively large process data sets from disciplined software processes to empirically characterize software defects in rigorous statistical terms that incorporate measures of dispersion as well as central tendency. The use of large data sets is unusual for the software industry; many published studies rely on fewer than 30 data points. The classroom data that I use for the majority of my analyses has about 10,000 observations, which allows me to use multiple data sets split by factors such as programming language. The richness of the data allows the use of statistical techniques that are not feasible with small data sets, such as sophisticated regression models and mixed models. Each observation has over 30 different

attributes, which are described in Appendix A along with the derived measures used in my analyses.

Real-time control of the software process, in the sense of making efficient day-to-day engineering and management decisions, depends on a realistic understanding of the defects injected and removed from the work product. Understanding the statistical nature of software defects is therefore crucial to the continuing maturation of the software industry because of the insight it provides into rework issues, which take as much as 50% of the effort in many projects [Krasner 1997]. An improved understanding of software defect patterns will help software engineers and managers make more informed and efficient decisions in controlling and improving the software process.

The number of defects injected into work products should intuitively be a function of the competence of the workers, the process used, the size and complexity of the work product, and previously injected defects in antecedent work products. Data from disciplined processes are analyzed to identify explanatory variables, as suggested by previous empirical research and by the drivers in widely used models for software projects.

Chapter 2 summarizes the published research relevant to analyzing the impact of process variables, programmer ability, and teamwork on software quality.

Chapter 3 describes the research methodology used to explore the software quality factors in a broad sense. It also discusses issues affecting the generalizability of my results and how the classroom data was cleaned up for these analyses.

Chapter 4 explores the factors that significantly affect software quality in the classroom. These factors can be broadly characterized as programmer ability, problem/solution complexity, technology issues, and process variables. Although some of these factors may be beyond the

control of a software manager, many – especially those related to process – can be influenced by engineering and managerial decisions. Potential confounding factors are investigated to determine which ones should be actively considered when analyzing the programmer and process variables that are of primary interest.

Chapter 5 considers how atypical data, which is not part of the common cause system for software development, can be effectively identified and excluded from analyses. Atypical data is frequently excluded from software research in an *ad hoc* manner. Systematically identifying data that may unduly influence results is preferred. It also enables a distinction between consistently performed processes (stable processes) and processes that conform to recommended best practices (capable processes). Additional techniques for identifying outliers based on regression models are considered in Chapter 7.

Chapter 6 investigates the statistical distributions that best describe software defects. Many of the statistical tools used in the software industry make distributional assumptions that are rarely verified. For example, u-charts are frequently used by organizations beginning to apply statistical process control to their software processes. The u-chart assumes a Poisson distribution that may not be empirically supported, suggesting that other techniques might be superior.

Chapter 7 contains multiple regression models and mixed models that predict software quality based on the process, people, technology, and product factors investigated in Chapters 4 to 6. Sophisticated statistical models capture quality factors and their interactions and focus attention on critical leverage points for process and people.

Chapter 8 expands the analysis of software quality factors beyond classroom data into project data from industry. Two projects are analyzed; one using processes directly derived from

those used in the classroom and one using processes that have systematically matured over many years. These two analyses provide an initial investigation into some of the team effects on software quality in the context of software inspections.

Chapter 9 summarizes the contributions of my research and identifies opportunities for future research based on my results.

The purpose of my research is not to build a defect prediction model. It is to characterize the contributions of process discipline to software quality for individuals and for team-based inspections. Defect prediction models need to be designed for, and calibrated to, the application domain, development environment, processes, and organizational culture of a software project.

The importance of my research lies in two conclusions. First, sophisticated statistical models using detailed process data are feasible and potentially useful, if appropriate techniques are used. This conclusion refutes those who argue that software processes are intrinsically too chaotic to benefit from statistical analysis and control. Second, even though performance depends on the capability of the people building the software, disciplined processes can significantly improve the performance of even the best workers. This conclusion refutes those who resist disciplined processes and prefer the "flexibility" of an *ad hoc* environment. A third conclusion is implied by my research and partially explored in Chapter 8: effective teams also significantly improve performance, specifically in the context of software inspections.

2.0 LITERATURE REVIEW

2.1 THE SOFTWARE PROCESS

A ***process*** is a sequence of steps performed for a given purpose [IEEE-610 1991]. The ***software process*** can be defined as a set of activities, methods, practices, and transformations that people use to develop and maintain software and the associated products (e.g., project plans, design documents, code, test cases, and user manuals) [Paulk et al. 1995, 8]. Any use of "software process" should emphasize the actions performed to achieve a given purpose rather than the description of the process, which may or may not be realized in practice.

The basic software life cycle can be described in terms of requirements analysis, design, coding, testing, operations, maintenance, and retirement of the software product. These processes may be part of a larger systems life cycle, subdivided, or composed into multiple iterations. For example, requirements analysis may be divided into requirements elicitation, feasibility studies, operational concept studies, and software requirements analysis; design may be divided into top-level design (architecture) and detailed design; and testing may be divided into unit, integration, system, and acceptance testing. Processes may be composed according to a variety of life cycle models, from the classic waterfall life cycle to an incremental or evolutionary life cycle [Davis 1997].

There are a number of models and standards for software process definition and improvement [Paulk 2001]. Among the best known and most widely used are the Capability

Maturity Model® for Software (Software CMM®), for improving organizational capability [Paulk et al. 1995]; the Personal Software Process℠ (PSP℠), for improving the capability of individuals [Humphrey 1995]; and the Team Software Process℠ (TSP℠), for improving the capability of teams [Humphrey 1999].

2.1.1 The Capability Maturity Model for Software

The Capability Maturity Model for Software is a five-level staged model for building organizational capability, which emphasizes quantitative management for controlling and improving the software process at Levels 4 and 5. A mature organization consistently implements mature processes in its software projects to achieve repeatable performance.

On the five-level CMM scale, Level 1 organizations follow an ad hoc process. They do whatever it takes to get the job done, relying on the competence and heroics of their staff for success. Their primary problems stem from poor management practices. Level 2 organizations have an effective project management system in place. They may not always make the right decision, but they have a framework for process consistency. Level 3 organizations have installed the infrastructure needed to support organizational learning across projects. They have common processes, training, and measures that support systematic process improvement.

Levels 4 and 5 in the CMM are based on applying quantitative techniques, particularly statistical techniques, to controlling and improving the software process. In SPC terms, Level 4 focuses on eliminating assignable causes of variation, and Level 5 addresses common causes of

---

® Capability Maturity Model and CMM are registered with the U.S. Patent and Trademark Office by Carnegie Mellon University.

℠ Personal Software Process, PSP, Team Software Process, and TSP are service marks of Carnegie Mellon University.

variation, although the Software CMM does not explicitly require the use of SPC or control charts. Levels 4 and 5 were originally described in terms of operational definitions and comparability in the presence of variation [Humphrey 1988]. Most high maturity organizations as assessed against the Software CMM use control charts and other rigorous statistical techniques [Paulk, Goldenson, and White 2000; Paulk and Chrissis 2000]. The effect of CMM-based process improvement on software quality as reported in several studies is summarized in Table 1.

Table 1 Software Quality at Different Software CMM Maturity Levels

Maturity Level	Delivered Defects / FP [Jones 1995]	Shipped Defects / KSLOC [Krasner 1990]	Relative Defect Density [Williams 1997]	Shipped Defects [Rifkin 1993]
5	0.05	0.5	0.05	1
4	0.14	2.5	0.1	5
3	0.27	3.5	0.2	7
2	0.44	6	0.4	12
1	0.75	30	1.0	61

2.1.2 The Personal Software Process

The Personal Software Process (PSP) applies the Software CMM concepts of process discipline and quantitative management to the work of the individual software professional in a classroom setting. PSP is taught as a one-semester university course at several universities or as a multi-week industry training course. It typically involves the development of ten programs, using increasingly sophisticated processes [Humphrey 1995]. The life cycle processes for PSP are planning, design, coding, compiling, testing, and a post-mortem activity for learning. The

primary development processes are design and coding, since there is no requirements analysis step.

There are four PSP major processes (PSP0, PSP1, PSP2, and PSP3), with minor variants for the first three (PSP0.1, PSP1.1, and PSP2.1; PSP3 can also be considered a minor variant of PSP2).

- **PSP0.** The "current" process of the student at the beginning of the course is PSP0. Basic measures of historical size, time, and defect data are collected to establish an initial baseline for assignment 1A (using a linked list, write a program to calculate the mean and standard deviation of a set of data). PSP0.1 adds a coding standard, process improvement proposals, and size measurement. It is used for assignments 2A (write a program to count program lines of code) and 3A (enhance 2A to count total program and object LOC.).
- **PSP1.** PSP1 adds size estimating and test reports. It is used for assignment 4A (using a linked list, write a program to calculate linear regression parameters). PSP1.1 adds task planning and schedule planning. It is used for assignments 5A (write a program to perform a numerical integration) and 6A (enhance 4A to calculate linear regression parameters and the prediction interval).
- **PSP2**. PSP2 introduces design reviews and code reviews. It is used for assignments 7A (using a linked list, write a program to calculate the correlation of two sets of data) and 8A (write a program to sort a linked list). PSP2.1 adds design templates. It is used for assignment 9A (using a linked list, write a program to do a χ^2 test for a normal distribution).

- **PSP3**. PSP3 introduces the concept of cyclic development – incrementally building a program in multiple cycles. It is used for assignment 10A (using a linked list, write a program to calculate the 3-parameter multiple regression parameters and the prediction interval).

Each level builds on the prior level by adding a few engineering or management activities. This minimizes the impact of process change on the engineer, who needs only to adapt the new techniques into an existing baseline of practices.

PSP students are asked to measure and record three basic types of data: time (effort), defects, and size. All other PSP measures are derived from these three basic measures. The information recorded for each defect includes the defect type, phase in which the defect was injected, phase in which it was removed, fix time, and a description of the problem and fix.

Lines of code (LOC) were chosen as the size measure for PSP because they can be automatically counted, precisely defined, and are well correlated with development effort. Size is also used to normalize other data, such as productivity (LOC per hour) and defect density (defects per KLOC, where K stands for "thousand") [Gill and Kemerer 1991; Withrow 1990]. Each PSP program involves some amount of new development, enhancement, and/or reuse. Developing new or modified code represents most of the programming effort in the PSP course; consequently, new and changed LOC is the basis for most size measurement in PSP.

Design and code reviews are introduced in assignment 7. Design and code reviews are personal reviews conducted by an engineer on his or her own design or code. They are designed to help engineers achieve 100% yield: all defects removed before compiling the program. Design templates are introduced in assignment 9. The design templates are for functional specifications, state specifications, logic specifications, and operational scenarios.

Over the course of the PSP assignments, studies have shown a decrease in defect density (and in its dispersion), which is replicated in PSP data collected for my research as is shown in Figure 1 [Hayes and Over 1997; Wesslen 2000]. Some outliers are trimmed in the box-and-whisker charts in the figures in this dissertation. Outliers can skew a statistical analysis, but they can also provide insight when appropriately investigated. In a retrospective study such as this, causal analysis of why outliers are atypical is not feasible. Discarding outliers without root cause analysis, however, can adversely affect the validity of conclusions. Interquartile limits, as illustrated by the "whiskers" in Figure 1, can be used to identify outliers (the limits are set at 1.5 times the interquartile range beyond the 25% and 75% quantiles [SAS Institute 2000, 36]). Consistent results of statistical analyses of the data both with and without the outliers suggest that the conclusions are robust.

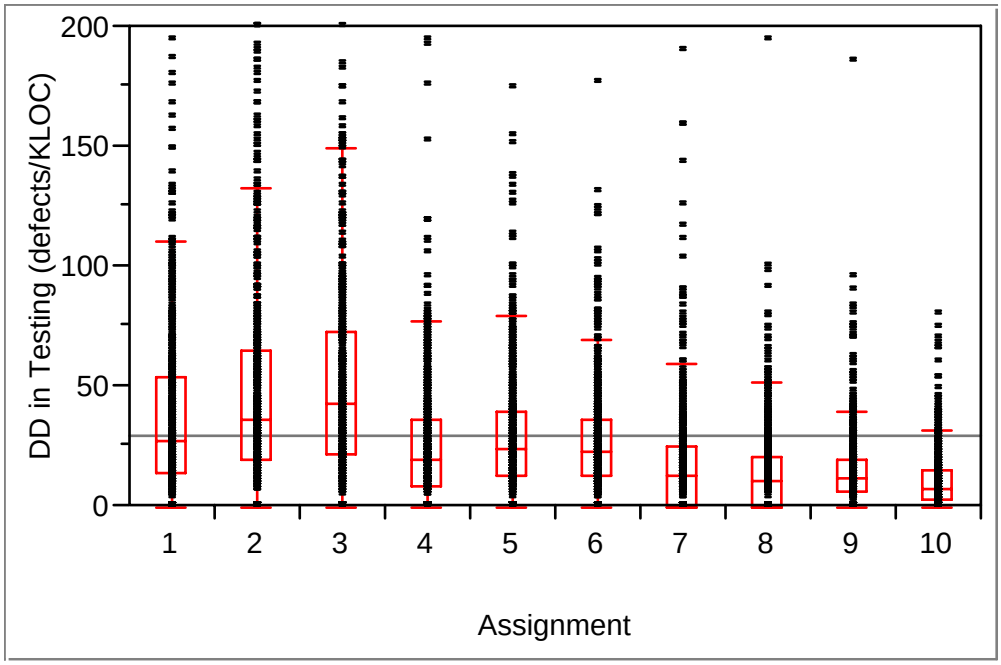

Figure 1 Trends in *Defect Density in Testing* Across PSP Assignments

Most PSP studies deal with improvement during the PSP course, showing percent decreases in *defect density in testing* between 63% and 82% [Humphrey 1996; Hayes and Over

1997; Wohlin and Wesslen 1998; Wesslen 2000], although one study addresses the impact across several companies before and after adopting PSP [Ferguson et al. 1997]. Some studies have observed that talented students do well regardless of how they structure their work time, but less-talented students benefit from a more disciplined approach [Hou and Tomayko 1998; Prechelt and Unger 2000].

It should be noted that quality improvement continues after the PSP class, as the PSP ideas are further internalized in an industry setting, and that software products developed by PSP-trained teams reportedly have very few to zero defects found in the field [Hayes 1998; Ferguson et al. 1997; Holmes 2003; Hirmanpour and Schofield 2003]. In spite of the positive impact of PSP, its continued use after the class depends on a working environment that actively supports its discipline [Prechelt and Unger 2000, 471].

2.1.3 The Team Software Process

The Team Software Process (TSP) brings PSP-trained students together as a team in an industry project setting. TSP is designed for use with teams of two to 20 members; a multi-team variant can be used for projects with up to 150 members. Companies that have adopted TSP have reported significant improvements in quality.

- Teradyne reported a 100X improvement over the company's typical projects [McAndrews 2000].
- The Taskview project at Hill Air Force Base reported an unprecedented performance, with only one high-priority defect found in candidate evaluation and system test, and reduction in test time from 22% of the project duration to 2.7% [Webb and Humphrey 1999; Webb 2000; McAndrews 2000].

- Boeing reported a 90% decrease in post-release defects and a decrease in test time of 94% [McAndrews 2000, 17].

2.1.4 Relevance of PSP, TSP, and CMM to My Research

These process improvement frameworks are relevant to my research because individuals and projects using these frameworks are performing disciplined, mature processes and are the sources of my data. They provide the context for exploring the impact of disciplined processes on software quality.

The bulk of my analyses use PSP data, although data from TSP and CMM high maturity projects are used in Chapter 8. Data from PSP, TSP, and high maturity CMM processes can be characterized as coming from consistently performed, disciplined processes that use appropriate techniques and tools. PSP and TSP are measurement-driven approaches, where the programmer's own data drives the learning process as incrementally more powerful processes are adopted. PSP in particular allows exploration of the impact of increasingly disciplined processes on quality, and the improvement in quality as the process matures can be empirically observed in PSP. Data from the three categories of disciplined processes give multiple perspectives on the effect of programmer ability and disciplined processes on software quality.

2.2 PEER REVIEWS

The most powerful defect detection technique for the work products of a software process is the peer review – reviews of work products by the peers of the producer to evaluate whether the conditions imposed on the work product at the start of the phase have been satisfied. Identification of defects in peer reviews is the primary mechanism for in-process control using

defect data. Reliance on identifying and repairing defects in testing is analogous to mass inspection in the manufacturing environment; "testing in quality" is notoriously ineffective and inefficient.

2.2.1 Inspections

A number of peer review methods have been defined, from informal walkthroughs to formal inspections [IEEE-1028 1988; Freedman and Weinberg 1990; Fagan 1976; Fagan 1986]. Peer reviews are reviews of a work product by the peers of its producer, which may be performed at any point during development, while testing occurs when an executable software module can be created, which is usually near the end of a development effort. It is generally accepted that inspections are the most effective peer review technique, with about five hours saved in testing for every hour spent in inspections [Ackerman, Buchwald, and Lewski 1989].

A typical set of rules for effective inspections includes the following:

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

The crucial point in understanding the power of peer reviews is 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. 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 [Boehm 1981]. Thus, in-process verification techniques such as inspections can have a significant impact on the cost, quality, and development time of the software since they can be applied early in the development cycle. It has also been observed, however, that no matter how well inspections are executed, they cannot overcome a seriously flawed development process [Weller 1993, 45].

Empirical research on the factors that lead to effective inspections has been contradictory, however, and it is unclear which factors are significant. Weller found that familiarity with the software product and preparation rate were the most important factors affecting inspection effectiveness, although for design inspections, controlling the amount of material inspected in a single meeting was also significant [Weller 1993]. Parnas and Weiss argue that a face-to-face meeting is ineffective and unnecessary [Parnas and Weiss 1987], even though in Fagan's inspection process, the meeting is where defect detection occurs (preparation time is for understanding the software product, not identifying defects). Supporting the argument against meetings, Eick and colleagues found that 90% of the defects could be found in preparation [Eick et al. 1998, 64]. The findings of other researchers also indicate that face-to-face meetings are of negligible value in finding defects [Porter and Johnson 1997; Perry et al. 2002; Land 2002]. Porter, Votta, and their colleagues conclude that, based on the competing views and conflicting arguments, we have yet to identify the fundamental drivers of inspection costs and benefits [Porter and Votta 1997; Perry et al. 2002].

Some software professionals disagree with this conclusion [Michael Fagan, personal communication, 13 April 2001], arguing that in many cases the studies were based on inspections that did not follow rules such as those listed earlier in this section. Fagan stated that two of the three essential requirements for implementing inspections were a proper description of the inspection process and its correct execution [Fagan 1986]. Identifying the drivers for effective inspections remains an active area of research, since the benefits of inspections are indisputable, even if alternative methods that might improve the inspection process remain shrouded in controversy.

2.2.2 Relevance of Peer Reviews to My Research

Peer reviews are relevant to my research because they are the source of the process data used to investigate the impact of disciplined processes on software quality. They instrument the software process; the data for meaningful analyses of quality would simply not be available without some form of peer review.

PSP reviews are a variant of peer review that invoke the formality of software inspections without actually involving peers in the review. This allows an investigation into the impact on software quality of a disciplined process on individual performance. These results can be contrasted to team effects in comparison to the TSP and high maturity peer reviews within the scope of what can be investigated in a retrospective analysis.

2.3 SOFTWARE QUALITY AND MEASUREMENT

The four general classes of measurement commonly used in the software industry are cost, schedule, functionality, and quality. Although cost is easily measured directly, effort in

terms of person-hours is a common surrogate for cost since it dominates the cost equation for software projects [Goethert, Bailey, and Busby 1992]. Schedules are usually measured in terms of calendar time and are predominantly determined by the effort needed to implement a given functionality and by resource allocation and conflict.

Size measures are typically used as surrogates for measuring functionality, e.g., the number of requirements [Abbot 1999], module count [Lehman et al. 1999], function points [Albrecht and Gaffney 1983; Kemerer 1993; Jones 1997], or lines of code (LOC) [Park 1992]. Size is also a useful leading indicator in management control, since mistakes in the size estimate can have dramatic ripple effects on planning parameters such as effort and schedule. Complexity metrics are closely correlated to size metrics like LOC [Fenton and Neil 1999; Graves et al. 2000].

When discussing "quality" in the software industry, "defects" is the common indicator [Florac 1992], although software quality characteristics include functionality, reliability, usability, efficiency, maintainability, and portability [ISO 9126]. When considering multiple dimensions of performance, whether explicitly labeled as "quality" or not, different practices are likely to affect different dimensions of performance [MacCormack et al. 2003]. Defect density is a common quality surrogate in the software industry [Gill and Kemerer 1991; Withrow 1990].

The total number of defects injected in a software product can never be identified with certainty. Even in high-reliability software, the possibility always exists that another defect lurks undiscovered. In operational terms, a cut-off point is identified for counting and comparing the total number of defects in a product. Common cut-off points for counting defects include at the end of acceptance testing, after six months of operational use in the field, and after one year of operational use. The choice of when to cut off data collection is constrained by the ability of the

organization to gather valid feedback from users and guided by the operational profiles of the users as the product is deployed. Defects are still identified and addressed as appropriate after the cut-off point, but the cut-off determines the data profile for purposes of comparison.

2.3.1 Characterizing Software Quality

The focus of my research is on software quality. A defect is a flaw in a system or system component that causes the system or component to fail to perform its required function. A defect, if encountered during execution, may cause a failure of the system. Defects may be categorized according to their expected severity, e.g., major/minor, or type, e.g., function, interface, or data [Chillarege and Bhandari 1992]. For in-process control during development, the defect data from peer reviews is a practical reliability surrogate.

The software process is a design-intensive, human-centric process. People naturally make mistakes as they design and build software. Reported defect injection rates range from 30 per KLOC [Boehm 1981] to 110 per KLOC [Hayes and Over 1997], depending on when defects are recorded and the operational definitions of "defect" and "line of code." Defect removal effectiveness for peer reviews ranges from 30% [Jones 1996] to over 90% [Fagan 1986; McCann 2001].

Defects have a significant impact on cost, effort, and schedule because of rework. It is not uncommon for software projects to run at 40-50% rework, although some high maturity organizations report rework under 5%, which implies shorter cycle times and higher customer satisfaction [Krasner 1997]. Research consistently finds that the Pareto principle (or 80-20 rule) applies to software: a small number of modules contain most of the defects discovered during pre-release testing, and a small number of modules contain most of the operational defects, e.g.,

87% of the defects found to lie in 26% of the modules [Cook and Roesch 1994; Schaefer 1985; Fenton and Ohlsson 2000; Khoshgoftaar et al. 1998].

In a survey of project managers, Schneberger identifies eight factors affecting software failures: 1) requirements change, addition, and definition; 2) programmer / team member experience and turnover; 3) design changes, scope, and complexity; 4) coding and testing phase problems; 5) new technology, languages, and tools; 6) ongoing experience; 7) upper management influence, bidding and time constraints; and 8) lack of data available to use in metrics and models [Schneberger 1997]. Evanco and Lacovava identified three factors affecting software failures: development complexity, percent of reusable code, and the experience and educational levels of the software development staff [Evanco and Lacovava 1994]. From a survey of 32 environmental factors affecting software reliability, Zhang found the top four factors to be program complexity, programmer skills, testing coverage, and testing effort [Zhang 1999]. MacCormack, Kemerer, Cusumano, and Crandall identified four factors affecting customer-reported defects: systems software projects, early prototypes, design reviews, and regression testing [MacCormack et al. 2003]. Neufelder identified ten process parameters whose presence affected fielded defect density; the most significant was “consistent and documented formal and informal reviews of the software and system requirements prior to design and code.” [Neufelder 2000]

In these studies, programmer experience and skills are frequently identified as important factors affecting software quality. Wohlin and Wesslen, however, found that previous experience and background have no statistically significant impact on defect density for PSP assignments [Wohlin and Wesslen 1998].

Many researchers have found a positive relationship between size and defects [Putnam and Myers 1997, 32; Criscione, Ferree, and Porter 2001]. Fenton and Neil found that LOC and complexity metrics are reasonable predictors of the absolute number of defects but very poor predictors of defect density; they concluded that complexity and/or size measures alone cannot provide accurate predictions of software defects [Fenton and Neil 1999]. Fenton and Ohlsson found that size metrics (such as LOC) are not good predictors of post-release defects in a module, a module's pre-release defect density, or a module's post-release defect density [Fenton and Ohlsson 2000].

Putnam and Myers found a log-log relationship between program size and the number of defects in testing. It is apparent from their results that programs must span more than four orders of magnitude in size before the relationship between size and quality becomes easily visible [Putnam and Myers 1997, 32].

Some researchers have found that defect density decreases as size increases [Basili and Perricone 1984; Shen et al. 1985], perhaps because the number of interface defects for smaller modules grows as the system becomes larger and more complex. Hatton approached this issue from a cognitive science perspective, arguing that those components that fit comfortably into short-term memory cache are the ones with the lowest defect densities and that components should be neither too small nor too large – the most reliable systems are those with component sizes grouped around 200-400 lines of code [Hatton 1997]. Gaffney also observed an optimum module size, although it was about 877 lines of code [Gaffney 1984].

2.3.2 Statistical Distribution of Defects

The distribution of defects may be viewed over the modules comprising a system or over time as they are discovered. Across modules, software defects are sometimes assumed to follow

a normal distribution [Raffo 1996]. They are sometimes assumed to follow a lognormal distribution, based on the fact that the product of random variables approaches a lognormal distribution and the observation that software data tend to be skewed [Raffo 1996; Mullen 1998; Hayes and Over 1997; Devnani-Chulani 1999, 86-87].

Over time, the discovery of software defects is frequently assumed to follow a (non-homogenous) Poisson process [Musa, Iannino, and Okumoto 1987, 255-259; Keiller 1995]. Non-homogenous implies that the parameters of the Poisson distribution change over time, i.e., defect injection and discovery are not uniform across life cycle phases. The times between detecting software defects are sometimes assumed to follow an exponential distribution (based on their being discovered according to an assumed Poisson distribution [Porter 2001; Wohlin and Runeson 1998, 402; Biffl 2000, 38]), a Rayleigh distribution (based on the logic that errors are proportional to work done, work done is proportional to effort, and effort follows a Rayleigh curve according to the SLIM cost model [Putnam and Myers 1997, 168]), or a gamma distribution [Graves et al. 2000, 657].

These statistical assumptions may be plausible in a particular context, but they are inconsistent and rarely verified empirically. It is sometimes even unclear whether the assumed distribution is over modules or over time. For my research, the distributions of interest are over modules.

2.3.3 Defect Prediction and Estimation Models

In reliability engineering, prediction models use parameters associated with the software product and its development environment, and estimation models apply statistical techniques to observed failures during testing and operation [AIAA R-013, 6]. Defect prediction models are useful earlier in the life cycle, which is beneficial from a management perspective. It is

commonly assumed that the software failure rate is related to the number of defects remaining in the software product [AIAA R-013, 18]. A number of defect prediction models have been built based on lines of code, number of decisions, number of subroutine calls, Halstead volume, McCabe cyclomatic complexity, and other measures [Lyu 1996; Fenton and Neil 1999].

Akiyama's model is based on program size [Akiyama 1972]:

$$D = 4.086 + 0.018\,L$$

where D is the expected number of defects found and L is lines of code. The data set consists of seven modules written in assembly language. For this size model, the correlation coefficient $\rho=0.83$. Akiyama also built a model based on the complexity of a program:

$$D = -0.084 + 0.12\,C$$

where C is the sum of the number of decision symbols and subroutine call symbols in the program's flow chart. For this complexity model, $\rho=0.92$.

Halstead's model is based on another size measure [Halstead 1977, 87-91]:

$$D = V / 3000$$

where the volume V is a size measure, which is the product of the total number of operators and operands times the $\log_2$ of the unique number of operators and operands. Using Akiyama's data, Halstead characterized this model as representing the data with considerable fidelity but does not report a statistical measure of goodness.

Compton and Withrow's model uses another of Halstead's size measures [Compton and Withrow 1990]:

$$D = 0.069 + 0.00156\,\hat{N} + 0.00000047\,\hat{N}^2$$

where $\hat{N}$ is Halstead's estimated program length, the sum of the unique operators and operands times their respective logs. When modeling all software packages, they found $R^2=0.0064$; but when modeling only those packages where defects were found, $R^2=0.9801$.

Jones' model uses a functionality-based size measure [Jones 1996]:

$$D = F^{1.25}$$

where F is the number of function points (a size measure). No measure of goodness was reported for this model, and Jones characterizes it as a useful rule of thumb rather than a rigorous model.

Lipow's model incorporates a technology factor for programming language along with program size [Lipow 1982]:

$$D / L = A_0 + A_1 (\ln L) + A_2 (\ln^2 L)$$

where the A_i coefficients are language-dependent factors. No measure of goodness was reported for this model.

Gaffney's model removes the technology factor [Gaffney 1984]:

$$D = 4.2 + 0.0015 L^{4/3}$$

It is a variant of Lipow's that assumes programming language is not a significant factor. The measure of goodness reported is *relative error=8.4%.*

Criscione, Ferree, and Porter's model is a process-based model that incorporates solution complexity and process effectiveness [Criscione, Ferree, and Porter 2001]:

$$D = (0.1)\ (0.2)\ (0.3)\ [0.4\ S_R] + (0.1)\ (0.2)\ [0.3\ S_D] + (0.1)\ [0.015\ S_C]$$

Data from previous phases in the product's life cycle is used to estimate test defects: summing the results from multiplying sizes of various work product times, empirical defect density (S_R, S_D, and S_C for requirements, design, and code), and the percentage of defects escaping a phase.

Their model depends on a stable development and testing environment [Fenton and Neil 1999, 677]. For four releases, the reported number of defects was 0.46, 0.47, 0.99, and 2.30 standard deviations from the predicted number using their model.

Takahashi and Kamayachi's model is a process-based model that incorporates problem complexity, programmer ability, and solution complexity [Takahashi and Kamayachi 1985]:

$$D = 67.98 + 0.4579\ SCHG - 9.687\ ISKL - 0.083\ DOCC$$

where *SCHG* is the frequency of program specification change, *ISKL* is the average number of years of programming experience for the team, and *DOCC* is the volume of program design documents. For this model, $R^2=0.6012$.

Defect prediction models based on simple measures of size and complexity do not consider the difficulty of the problem, the complexity of the proposed solution, the skill of the programmer, or the software engineering techniques used [Fenton and Neil 1999, 683]. Multivariate analyses, such as factor analysis, generate synthetic measures that combine a number of different measures, such as lines of code and complexity, to avoid multicollinearity, but the practical application of the synthetic metric may be obscure.

One of the more detailed defect prediction models is the COnstructive QUALity MOdel (COQUALMO), which has defect injection and removal submodels that incorporate 21 of the COCOMO II cost drivers, with the exception of *development flexibility* [Boehm et al. 2000, 254-268; Devnani-Chulani 1999]. COCOMO II has 17 multiplicative cost drivers (or effort multipliers), which are grouped into four categories, and five scaling cost drivers, as listed in Table 2. Each cost driver can accept one of six possible qualitative ratings, ranging from very low to extra high.

Table 2 COCOMO II and COQUALMO Drivers

COCOMO II Categories	COCOMO II and COQUALMO Drivers
Product factors	Required software reliability Data base size Product complexity Required reusability Documentation match to life cycle needs
Platform factors	Execution time constraint Main storage constraint Platform volatility
Personnel factors	Analyst capability Programmer capability Applications experience Platform experience Language and tool experience Personnel continuity
Project factors	Use of software tools Multi-site development Required development schedule
Scaling cost drivers	Precedentedness Development flexibility *(not in COQUALMO)* Architecture and risk resolution Team cohesion Process maturity

The Software Error Estimation Reporter (STEER), which is based on SLIM, characterizes defect patterns over time by a Rayleigh curve [Kan 1995, 191-192]. Capture recapture models, originally developed for estimating wildlife populations, are based on extrapolating from defects found by multiple inspectors [Briand et al. 2000; El Emam and

Laitenberger 2001; Humphrey 1999, 245-250]. Comparisons of different defect prediction models suggest that different models are best for different environments; it is not possible to find a single superior model [Brocklehurst and Littlewood 1996, 126-127; Wohlin and Runeson 1998, 407-408].

Inconsistent or undocumented decisions about when in a product's life cycle to stop counting defects, e.g., at acceptance test, six months after delivery, one year after delivery, etc., make it difficult to determine whether a model is predicting discovered or residual defects.

Fenton and Neil have observed that current approaches to defect prediction must deal with several as yet unresolved issues. Some issues are based on poor operational definitions or problems in statistical methodology, some depend on the unknown relationship between defects and failures, and some are intrinsic problems with using size and complexity metrics as the (sole) predictors of defects. They conclude that traditional statistical (regression-based) methods are inappropriate for defect prediction, preferring Bayesian techniques, and that more complete models should include other explanatory factors, such as testing effort and operational usage [Fenton and Neil 1999, 153].

2.3.4 Relevance of Defect Prediction Models to My Research

Defect prediction models are relevant to my research because the simple regression models in Chapter 4 and the multiple regression models and mixed models in Chapter 7 are defect prediction models. The PSP data provides an abundance of process and contextual information that can be investigated for their impact on software quality; this addresses the desire for "more complete models" expressed by Fenton and Neil [Fenton and Neil 1999, 153]. Although the PSP environment is too limited for general use (other factors may also be important in a team context), the objective of building the PSP models is to identify the factors that are

important determinants of software quality and to quantify their relative contributions. Managers and engineers can then make informed decisions about software processes with an understanding of the relative impact of those decisions on software quality.

Some of the questions posed by researchers such as Fenton and Neil are addressed by my models, e.g., the feasibility of more comprehensive process-based models than have been built by previous researchers. The previous models contributed to identifying factors that were considered, where feasible, in building my defect prediction models.

2.4 DIFFERENCES IN INDIVIDUAL PERFORMANCE

Differences in performance between individuals, which may span an order of magnitude, are generally acknowledged to be the greatest source of variability in software engineering research [Hayes and Over 1997, 22; Wohlin 2004, 212]. Weinberg observed, "Individual variation is… the bane of project predictability. The social nature of team programming can be used to average out this variation – but such averaging prevents us from getting experimental information on individual programmers. The devastating cost of individual variation on real projects has supported the validity of my prediction that the individual working alone is neither a fruitful unit of study, nor a productive component of programming project work" [Weinberg 1998, 3.iii].

2.4.1 Order of Magnitude Differences

Curtis observes that many software technology advances will be masked by the impact of individual differences [Curtis 1988, 279]. The earliest known study of differences in performance between professional programmers by Sackman and colleagues found a 28:1

difference [Sackman, Erikson, and Grant 1968, 6]. McGarry found a performance difference of 22:1 on small projects (less than 20 KSLOC) and of 8:1 on larger projects [McGarry 1982, 226].

Perhaps the best known example of the variation in individual performance is a study by DeMarco and Lister of productivity in programming, which showed the best people outperforming the worst by about 10:1, the best outperforming the median performer by about 2.5:1, and the half that were better than the median outperforming the half that were worse than the median by more than 2:1 [DeMarco and Lister 1999, 45]. Even in controlled experiments, the variation related to individual differences accounted for one-third to one-half of the variation in performance [Curtis 1988, 286].

2.4.2 Relevance of Individual Differences to My Research

Individual differences are important to my research because much of the resistance to disciplined processes lies in the fear that discipline will cripple creativity and agility [Highsmith 2000, 11-13; Boehm and Turner 2004, 1-24]. Process discipline and powerful statistical techniques enable us to address Weinberg's concerns over the devastating impact of individual variation. The importance of balancing discipline and creativity, however, is well stated by Glass: "If we appreciate science, we understand that the discipline imposed by the scientific mind forms a frame for the opportunistic, even serendipitous, and certainly creative discoveries that constitute an amazing portion of what science has given us" [Glass 1995, 41]. Statistical evidence that even top professionals can significantly improve both their performance and their consistency by appropriately implementing disciplined processes can help sway resisters to consider more disciplined approaches.

2.5 TEAM PERFORMANCE

Statisticians have observed that "group thinking is usually better, less variable, and more precise than individual thinking" [Hare et al. 1995, 54] There is empirical support for team performance typically surpassing that of individuals in a wide range of problem solving tasks [Brodbeck and Greitemeyer 2000, 621; Morgan and Tindale 2002, 46; Land 2000, 181-182]. DeMarco and Lister have observed that the range of team performance, rather than being an order of magnitude, tends to be between 85% and 115% of the norm, after removing other risk factors [DeMarco and Lister 2003]. A meta-analysis of 27 independent studies indicates that 81% of the increase of performance of groups over individuals is due to statistical pooling of the participants and 19% of the improvement is due to interaction effects within the group [Kramer 1998, 23]. It is also worth noting that high academic performers tend to collaborate more than low performers [Land 2000, 82].

Software researchers have focused on the qualitative aspects of team building [Scholtes 1996] and workgroup development [Curtis, Hefley, and Miller 2001, 285-308] rather than quantifying the differences in performance between individuals and teams in various contexts, although Curtis notes that the productivity differences between high-performance and low-performance teams are typically about 3:1 [Curtis 1988, 288]. Research has focused on how to structure and build high performance teams, e.g., chief programmer teams [Baker 1972; Baker and Mills 1973] and structured open teams [Constantine 1995, 83-86].

2.5.1 Inspection Teams

One of the few quantitative analyses of software team effectiveness deals with the size of an inspection team. The recommended size is four participants [Fagan 1976, 191], although

three to five is usually considered acceptable [Briand et al. 2000, 519; Bourgeois 1996]. Some researchers identify four as the most effective team size, followed by team sizes of three and then five [Mah 2001, 12; Hall and Nixon 2000, 17]. Others simply recommend "more than two" [Glass 1999, 19]. In contrast, some researchers argue that there is no significant difference in effectiveness between teams of size two and teams of size four [Perry et al. 2002, 697]. Related research on pair programming suggests that pairs of programmers working together have more consistent and superior results than individuals working alone [Williams et al. 2000, 23; Williams 2000, 40]. Recommended inspection team sizes therefore range from two to five, with the caveat that relevant expertise be represented on the team.

In general, team size and composition is known to affect performance: teams composed exclusively of low-ability individuals show process loss, and large teams can result in social loafing and diffusion of responsibility [Bowers, Pharmer, and Salas 2000, 310-314]. There may be an optimal inspection team size, but factors such as the inspection process structure, techniques, inputs, context, and technology are also critical and likely to affect the best team size [Porter and Votta 1997].

2.5.2 Relevance of Team Performance to My Research

Team performance is relevant to my research because effective teams are considered a prerequisite for applying statistical control to software processes. While the bulk of my research focuses on individual performance in PSP, the statistical models built highlight the fact that there are order-of-magnitude differences between individuals. Disciplined processes improve performance and lessen variation, but much more is necessary before statistical control is truly feasible. Effective teams improve performance and lessen variation over and above the contribution of disciplined processes, as explored in Chapter 8.

2.6 STATISTICAL THINKING

Statistics deals with the collection and analysis of data to solve real-world problems in the presence of variability [Hogg and Ledolter 1992, 1]. The fundamental axioms of statistical thinking are that all work is a series of interconnected processes, all processes are variable, and understanding the impact of variation leads to better decisions and systematic improvement [Britz et al. 1997; Hare et al. 1995]. These axioms embody a way of thinking, a way of acting, and a way of understanding the data generated by processes that collectively result in improved quality, increased productivity and competitive products.

Statistical thinking is fundamental to TQM [Deming 1986; Hogg and Ledolter 1992, 7-8]. Controlled processes are stable, and stable processes are predictable. If a controlled process is not capable of meeting customer requirements or other business objectives, the process must be improved or retargeted.

The statistical thinking characteristic of a high maturity organization depends on two fundamental principles. First, process data is collected at the "process step" level for real-time process control. Engineers use data to drive technical decision making in real-time, thereby maximizing efficiency. Second, and a direct consequence of statistical thinking, is that decision making incorporates an understanding of variation.

The statistical distributions described in Table 3 have all been suggested as appropriate for describing software defect patterns, either over time or over modules. The normal distribution is frequently assumed simply because it is common [Hogg and Ledolter 1992, 19-24]. The lognormal distribution is frequently assumed because the product of random variables tends towards the lognormal [Hogg and Ledolter 1992, 131-133]. Putnam and colleagues have

found that many software processes follow a Rayleigh distribution over time [Putnam and Myers 1992, 45-46; Montgomery 1996, 67; Leemis 1995, 88-89]. Defect data is frequently assumed to follow the Poisson distribution [Hogg and Ledolter 1992, 102-104], although the negative binomial distribution has been found more appropriate by Das [Das 2003; Montgomery and Runger 1999, 124-126; Williamson and Bretherton 1963, 7-10]. The normal, lognormal, and Rayleigh distributions are continuous. The Poisson and negative binomial distributions are discrete.

Table 3 Statistical Distributions Used to Describe Software Defects

Distribution	**Probability Density Function f(x)**	**Mean**	**Variance**
Normal, $N(\mu,\sigma^2)$	$\frac{1}{\sqrt{2\pi}\sigma} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$	μ	σ^2
Lognormal, $\Lambda(\mu,\sigma^2)$	$\frac{1}{\sqrt{2\pi}\sigma x} e^{-\frac{(\ln x-\mu)^2}{2\sigma^2}}$ $0 \le x < +\infty$	$e^{(\mu+\sigma^2)}$	$e^{2\mu+\sigma^2} (e^{\sigma^2}-1)$
Rayleigh	$2\lambda^2 t e^{-(\lambda t)^2}$ $0 \le t < +\infty$	$\frac{1}{\lambda}\Gamma(\frac{1}{2})$	$\frac{1}{\lambda^2}[\Gamma(2)-\{\frac{1}{2}\Gamma(\frac{1}{2})\}^2]$
Poisson	$\frac{\lambda^x e^{-\lambda}}{x!}$ x integer, $0 \le x < +\infty$	λ	λ
Negative binomial	$\binom{n-1}{k-1}(1-p)^{n-k} p^k$	$\frac{k(1-p)}{p}$	$\frac{k(1-p)}{p^2}$

The mean of a distribution is typically denoted with the Greek letter μ, and the variance is denoted with σ^2. If a random variable *Y* is lognormal, then $X=ln(Y)$ is $N(\mu,\sigma^2)$. For the Rayleigh distribution, λ is a shape parameter and *t* is time. For the Poisson distribution, the mean is equal to the variance, and λ is the single parameter expressing both. For the negative binomial

distribution, given a constant probability *p* of a success (or failure), the random variable indicates the number of trials until *k* successes (or failures) occur.

2.6.1 Operational Definitions

The first step in establishing a statistical understanding of a process is understanding how it is measured so that process consistency and data validity provide a basis for rigorous analysis. When looking at their process data, software organizations typically discover that measures are not as consistently defined and data are not as consistently collected as desired. Software organizations also typically discover that the defined processes used by the projects are not as consistently implemented as desired.

Well-defined processes are a prerequisite for statistical process control since consistent process performance is necessary for quantitative or statistical management. Wheeler expresses this point from an SPC perspective as "improvement begins with establishing operational definitions and standardizing procedures" [Wheeler and Poling 1998, 270]. Much of the work in software measurement has been aimed at building frameworks for establishing good operational definitions for such fundamental measures as effort [Goethert, Bailey, and Busby 1992], lines of code [Park 1992], and defects [Florac 1992].

2.6.2 Process Behavior and Control Charts

The questions of process consistency, effectiveness, and efficiency require measurement of process behavior as it is being executed over some reasonable time period. Other disciplines have addressed this issue by using statistical process control methods.

Statistical process control (SPC) can be defined as the use of statistical techniques and tools to analyze a process or its outputs to control, manage, and improve the quality of the output

or the capability of the process. Operationally, statistical process control implies the use of seven basic tools: flow charts, scatter diagrams, histograms, Pareto analysis, cause-and-effect (fishbone) diagrams, run (trend) charts, and control charts [Ishikawa 1986]. It is generally accepted, however, that SPC implies the use of control charts. Control charts provide a sound foundation for making process decisions and predicting process behavior [Wheeler and Chambers 1992].

A control chart is a run chart with upper (UCL) and lower control limits (LCL) added that indicate the normal execution of the process. The control limits are normally based on $\pm 3\sigma$ boundaries for the underlying common cause system that the process represents. Control charts provide a statistical method for distinguishing between variation caused by normal process operation and variation caused by anomalies in the process.

Common cause variation is variation in process performance due to normal or inherent interaction among the system components (people, processes, machines, material, and environment). It is characterized by a stable and consistent pattern over time. This variation is random and will vary within predictable bounds. The 3σ limits identify the amount of intrinsic variation that is natural to the process. This is the "voice of the process" telling what it is capable of doing. If this performance is satisfactory, the process is "capable." If the predictable performance is not satisfactory, then the process must be changed if the requirements for the process are to be satisfied since the variation is intrinsic to the common cause system underlying the process data.

Assignable cause variation, or special cause variation, is caused by anomalies that have marked impact on product characteristics and other measures of process performance. Assignable cause variations arise from events that are not part of the normal process. They

represent sudden or persistent abnormal changes to one or more of the process components. These changes can be in things such as inputs to the process, the environment, the process steps themselves, or the way in which the process steps are executed. When all assignable causes have been removed and prevented from recurring in the future so that only a single, constant system of chance causes remains, the process is stable (predictable), and unexpected results are rare.

Stability of a process with respect to any given attribute is determined by measuring the attribute and tracking the results over time. If one or more measurements fall outside the range of chance variation, or if systematic patterns are apparent, the process may not be stable, and a causal analysis should be performed. When all assignable causes have been removed and prevented from recurring in the future so that only a single, constant system of chance causes remains, the process is stable and predictable.

The simplest rule for detecting a signal (a possible assignable cause) is when a point falls outside the 3σ control limits. Many other sets of detection rules have been proposed [Wheeler and Chambers 1992, 96], which both make the control chart more sensitive to signals and also lead to a greater number of false alarms. The decision on which detection rules to use should be based on the economic trade-off between sensitivity and unnecessary work.

When process performance falls outside of the 3σ limits, the variation is very likely caused by an anomaly in the process. Shewhart used Tchebycheff's theorem to put a bound of 11% of the data being outside 3σ limits for any data set, thus identifying the worst-case boundary for false alarms when using 3σ limits [Shewhart 1939, 91]. Wheeler's Empirical Rule characterizes the typical behavior of a homogenous data set as having approximately 99-100% of the data within 3σ of the average [Wheeler and Chambers 1992, 61].

There are many different kinds of control chart. The XbarR chart was the first developed [Shewhart 1931]. It plots the averages of a homogeneous subgroup of data, e.g., for a sample taken from an assembly line at a particular point in time, on an Xbar chart (the "bar" indicates an average of the attribute of interest for the subgroup). The variation within the subgroup is captured on an R (or range) chart.

For many software processes, it is more desirable to plot individual data points than the averages of subgroups. The most commonly used chart for individual data is the XmR chart [Wheeler and Poling 1998], also known as the individuals and moving range chart, although other charts that take advantage of knowledge about the statistical distribution of the data can also be used when appropriate. Two graphs are generated in an XmR chart: an X chart for the individual values and an mR chart for the moving ranges, i.e., $mR_i = |X_i - X_{i-1}|$. The upper and lower control limits for the X chart (UCL_X and LCL_X), and the upper control limit for the mR chart (UCL_R), are calculated by

$$UCL_x = \overline{X} + 2.66(\overline{mR})$$

$$LCL_x = \overline{X} - 2.66(\overline{mR})$$

$$UCL_R = 3.268(\overline{mR})$$

where the X's are the individual values and the mR's are the moving ranges between adjacent values. The lower control limit for the moving range chart is always zero. $\overline{X}$ is the average of the individual values, and $\overline{mR}$ is the average of the moving ranges.

Control charts and other statistical tools can be ineffective if operational definitions are poorly formulated or aggregated data is used. Aggregated data has elements that are combinations (mixtures) of values from non-homogeneous sources. When initially looking at their process data, software organizations typically discover that the defined processes used by

the projects are not as consistently implemented or measured as believed. This initial investigation, sometimes called "informally stabilizing the process," involves understanding and refining the operational definitions of processes and measures and categorizing the processes/data into reasonably homogenous sets.

The control charts making the minimal assumptions about the underlying process are XbarR and XmR charts [Wheeler 2000]. Other control charts, such as the u-chart, make distributional assumptions about the data, e.g., the u-chart assumes the data follow a Poisson distribution. The most commonly used control charts by high maturity software organizations are XmR charts and u-charts [Paulk, Goldenson, and White 2000, 58-59]. It is important to verify that distributional assumptions are appropriate. As Wheeler expresses this issue, "If the theory is right, then the theoretical value is right, and the empirical value will mimic the theoretical value. But if the theory is wrong, then the theoretical value will be wrong, yet the empirical value will still be correct" [Wheeler and Poling 1998, 184].

2.6.3 Applying Statistical Control to Software Processes

From an industrial engineering perspective, SPC procedures are by default part of the minimum essential information needed to fully describe a process [Manley 1998, 221-230]. This is not a normal assumption in the software industry. Some doubt that SPC can be applied to software processes [Ould 1996; Kan 1995, 143-144]. Some of the objections are based on misunderstandings, e.g. the argument that data has to be normally distributed to apply SPC [Ould 1996]. Valid concerns center on the intrinsic high variability of software work, the validity of considering the combined output of multiple individuals on a single control chart, and the potential for causing dysfunctional behavior by the motivational use of data.

One of the first concerns about process performance is compliance: is the process being executed properly, are the personnel trained, are appropriate tools available, etc. If the process is not in compliance, there is little chance of performing consistently or satisfactorily. Even if the process is consistently performed, the intrinsic variation may be so great that no value can be obtained from statistical analysis; individual differences in performance can span an order of magnitude [DeMarco and Lister 1999, 45; Curtis 1981].

One source of variation is that items expected to be on the same control chart, e.g., different code modules, may be produced by different members of the team [Mayer and Sykes 1992, 212]. The result is software process data that is aggregated across individuals. In a manufacturing environment, placing data from different machines on the same control chart is not recommended.

Collecting software data on an individual basis would address this, but could have severe consequences if there were any chance of motivational use of the data, e.g., during performance appraisals. Deming was a strong advocate of statistical techniques and strongly averse to performance evaluations, declaring performance measurement "the most powerful inhibitor to quality and productivity in the Western world" [Deming 1986]. Austin has shown that the potential for dysfunction arises when any critical dimension of effort expenditure is not measured, and unless the latitude to subvert measures can be eliminated, i.e, measures can be made perfect, or a means established for preventing the motivational use of data, dysfunction is destined to accompany organizational measurement [Austin 1996].

Analyzing data at the individual level would also significantly decrease the amount of the data available for any specific statistical analysis at the team, project, or organizational level. Disaggregating process data by individual, by defect type, or by other categories may be critical

to obtaining insight into separate common cause systems, but this may imply severe practical limits to the value of SPC for software processes [Florac and Carleton 1999; Florac, Carleton, and Barnard 2000; Wheeler and Poling 1998, 270].

On the other hand, processes and systems are subject to hundreds of cause-and-effect relationships [Wheeler and Poling 1998, 85]. When every process is subject to many different cause-and-effect relationships, predictable processes are those where the net effect of the multiple causes is in a sort of equilibrium, which can be characterized as the common cause system [Wheeler and Poling 1998, 87]. Pyzdek comments that even companies producing one-of-a-kind products usually do so with the same equipment, employees, and facilities, and the key to controlling the quality of single parts is to concentrate on process elements rather than on product features [Pyzdek 1993, 53]. Each software product is unique, but is generated using a potentially repeatable process.

It is generally acknowledged that there are three conditions expected before design-intensive processes can be seriously considered for statistical control: best practices, discipline, and teamwork. Software engineering that uses "best practices" supports repeatable performance [Fagan 1986]. Less effective practices generally increase variation; for example, the only form of peer review successfully used for SPC is inspections, the most formal variant. A disciplined process with consistent performance will have less variability than an *ad hoc* process [Hayes and Over 1997, 34; Wesslen 2000, 113]. Inconsistently performed processes are by definition unstable. An effective team will demonstrate less variability than that shown by individuals [Hare et al. 1995, 54].

Applying SPC even to the work of effective teams applying disciplined processes may result in significant work to "informally stabilize the process" as process consistency and

disaggregation issues are identified and addressed. This investment has been deemed worthwhile by a number of CMM high maturity organizations that are obtaining business value by applying SPC to their software processes [Paulk, Goldenson, and White 2000; Paulk and Chrissis 2000; Florac, Carleton, and Barnard 2000; Weller 2000]. This suggests a useful degree of predictability is possible, even if significant variation remains in the common cause system.

2.6.4 Relevance of Statistical Thinking to My Research

Statistical thinking is relevant to my research because my research is an exemplar of applying statistical thinking to software processes. Basic statistical concepts such as good operational definitions and well-instrumented processes such as PSP are prerequisites for empirical research.

The amount of variation due to individual differences makes management by fact difficult, but identifying the common cause systems in PSP enables me to determine whether disciplined processes result in better performance and less variation than *ad hoc* processes. More importantly, analysis of PSP data allows a quantitative answer as to how much better disciplined processes are than *ad hoc* processes, and how much less variation they have.

2.7 SUMMARIZING THE RELEVANCE OF PRIOR RESEARCH

My research primarily relies on analyzing PSP data, which supports statistical research into factors that affect software quality. The PSP data set is both comprehensive and well-defined. It supports investigation of people, technology, and process factors, and the granularity of its process measurement allows a much more exhaustive study of process variables than is typically possible for software projects. Because the PSP data is for individual professionals, an

investigation of the effect of *programmer ability* is possible in concert with many attributes that characterize the individual programmers, e.g., *years of experience*. The large size of the PSP data sets allows the use of sophisticated and powerful statistical techniques, e.g., mixed models.

The focus of my analysis of PSP are the review processes, similar to inspection, for design and code. Other factors identified in research on defect prediction are considered where relevant. Because PSP data is for individuals rather than teams, the high variability in individual performance is a concern, but it is alleviated by the richness of the PSP data and the use of statistical techniques, such as outlier identification for removing extreme observations and mixed models for addressing individual differences.

Analyses of TSP and high maturity projects provide some additional insight into factors affecting the defect removal effectiveness of inspections. Although the data sets are relatively small and not as comprehensive as the PSP data sets, they are adequate for an initial exploration of team/project issues.

My research therefore applies statistical techniques, including regression models, analysis of variance, and mixed models, to understanding the relationship of process measures and other factors to software quality. It examines data from disciplined software processes as defined by PSP, TSP, and the Software CMM, allowing an exploration of both individual and team factors. Control charts, interquartile limits, and regression diagnostics are used to identify atypical programs. The appropriateness of various assumptions about statistical distributions is tested. In the end, the importance of both competent professionals and disciplined processes in building high-quality software is confirmed and quantified, which should aid software professionals and managers in implementing good software engineering practices in a turbulent world.

3.0 RESEARCH METHODOLOGY

3.1 THE RESEARCH QUESTIONS

Given a disciplined software process performed by an effective team, the question to explore is whether useful insights can be provided to software managers and engineers on the quality of the work products, expressed in defects, using statistical techniques. In this dissertation, I address the following questions; the answers will help managers and engineers plan their work, choose appropriate statistical tools, and efficiently control their processes.

- What are the factors associated with process discipline and programmer ability that impact software quality?
- Are the control limits for disciplined processes within the specification limits for good software engineering practice, specifically with respect to peer reviews?
- Can defect data for software design, coding, and testing be reasonably described by statistical distributions such as Poisson and lognormal?
- Can a useful defect prediction model be built, using the factors identified earlier?
- What factors, such as team size, affect the defect removal effectiveness of inspections?

The first two analyses deal with the impact of a disciplined process on software quality. In the first analysis, the impact of programmer ability is a crucial factor that must be identified and separated from that of a disciplined process. While project managers may not be able to exercise the control they might like on the ability of their staff, they have direct control over the

engineering discipline applied. If control charts do not add insight beyond what good engineering practice would suggest, i.e., the control limits are outside the bounds set by recommended practice, then their value is negligible in this context.

The third and fourth analyses deal with a statistical understanding of software defects, given a disciplined process. Such an understanding is useful for process modeling and training tools.

The fifth analysis focuses on the impact of effective teams on software quality in the specific context of peer reviews. It may provide useful recommendations for performing peer reviews, specifically for decisions on how they should be performed.

3.2 RETROSPECTIVE DATA SETS

My research uses retrospective data from PSP, TSP, and CMM high maturity projects. These data are used for exploratory, observational studies. Sufficient data has been collected, particularly for the PSP data, that the rule of thumb that there should be six to ten cases for every potential explanatory variable is easily satisfied in most instances [Neter et al. 1996, 330].

The primary data set used in investigating these issues is PSP class data. There are several advantages to using PSP data. First, disciplined processes are followed in the ending assignments, 7A to 10A. Second, many potential explanatory variables, such as those associated with teams or requirements volatility, that might confound the analysis can be eliminated from consideration. Third, a reasonably comprehensive, detailed, and large data set is available from PSP classes since 1993.

One set of PSP data, covering PSP classes from 1993 to 1996, was previously used in analyzing the results of PSP, and the results of my research can be compared to that report

[Hayes and Over 1997], although with the caveat that the objectives of the two studies are different. Hayes and Over were interested in the impact of PSP on estimation accuracy, productivity, and quality; my research focuses on the underlying process drivers that affect quality, such as review rates, which are independent of whether the PSP processes are the encompassing framework. This first data set, labeled PSPa for the 1997 report, contains data for 2,365 assignments and 298 students. The second PSP data set covers PSP classes from 1994 to 2001. This PSPb data set contains data for 10,347 assignments and 1,345 students.

Using the PSP data for understanding the effect of disciplined processes allows an inference of causality by meeting three methodological conditions: 1) the presumed cause and effect are related, 2) the presumed cause precedes the effect in time, and 3) other competing explanations for the observed effect can be ruled out [Duncan et al. 1999, 1]. The focus of the PSP class is on instilling disciplined processes, not learning programming languages or other techniques, therefore the predominant cause of any effects observed in the PSP data should be driven by increased discipline. The observed effects follow the process changes instilled by PSP. The PSP environment is sufficiently controlled that competing explanations should not have a significant impact, although they will be considered in my research so they can be ruled out explicitly.

Outliers can skew a statistical analysis, but they can also provide insight when appropriately investigated. In a retrospective study such as this, causal analysis of why outliers are atypical is not feasible. Discarding outliers without root cause analysis, however, can adversely affect the validity of conclusions. Interquartile limits can be used to identify outliers (the limits are set at 1.5 times the interquartile range beyond the 25% and 75% quantiles).

Consistent results of statistical analyses of the data both with and without the outliers suggest that the conclusions are robust.

Software measures, such as the number of defects, are usually normalized by the size of the program in lines of code to provide a defect density. During development, however, the number of lines of code may not be known, so other measures may be used such as function points, number of requirements, or number of pages of design [Abbott 1999]. Since this is a retrospective analysis, the actual software size can be used as the "area of opportunity" for normalizing the data, although only an estimate of LOC would be available during the early life cycle phase when requirements and design work products are built and inspected. In the case of the PSP data, residual defects after testing remain undiscovered; the cut-off point is when the assignment is turned in.

The primary advantage in using TSP data is that team members follow the PSP processes and collect PSP data, plus TSP data, therefore differences in operational definitions are minimized. TSP data can be easily compared to PSP data, and team performance can be compared to individual performance (including variation) with minimal issues with respect to differing operational definitions. Concerns in using TSP data center around explanatory variables that are introduced, such as requirements volatility, which need not be considered in the PSP environment.

In general, data from CMM high maturity projects may be well-defined within an organization, but operational definitions may differ significantly between the projects. While comparisons between PSP and TSP data are not unreasonable, comparing results from different organizations directly is inadvisable, although analysis of the factors affecting those results, where comparable data is available, is feasible.

3.3 OVERVIEW OF THE ANALYSIS PROCESS

My research is based on a series of observational studies rather than a controlled experiment. These are confirmatory studies in the sense that most of the factors considered for affecting quality have been previously studied. The inconsistencies in the results of previous studies may have been due to deficiencies in the data collected, small data sets, inconsistencies in process fidelity, or problems in generalizing and comparing results.

The use of data from high maturity environments implies that a reasonably comprehensive set of measures has been identified and that a defined process has been consistently implemented. This data should have relatively few confounding factors for investigating plausible explanatory variables, such as *years of experience*, that may affect software quality.

The first step is to collect the data, which should include the size of each module, the number of inspectors per module, inspection preparation time, inspection meeting time, and the number of defects injected and detected by life cycle phase. For PSP data, individuals review their own work, so there is no inspection team, and preparation and review times are synonymous.

The second step is to remove invalid data. An example of invalid data is where the number of defects removed in an inspection is greater than the number injected at that point in time. For PSP data, the students perform a causal analysis of every defect found and where it was injected as well as where it was found. This allows verification of internal data reliability. This is described in Section 3.5.

The third step is to identify explanatory variables that affect software quality. PSP data on potential explanatory variables, such as *programming language* used and *years of experience*, are available for many of the students, although a full set of demographic data was not consistently collected for all classes. Although data on non-process variables were not consistently captured, process variables, such as time spent in different activities and the number of defects found in reviews, were consistently recorded. This is described in Chapter 4.

If a variable does not affect software quality in a statistically significant way, then it can be ignored. If a variable is statistically significant, but its contribution is minimal in practical terms, then it can be ignored. If it is statistically and practically significant, it must be addressed in the further steps of the analysis. One way of addressing such a variable is to simplify it out of the analysis, e.g., if *programming language* is a significant explanatory variable, disaggregate the data by language and analyze modules written in a single language. Some variables may need to be transformed, since non-linear relationships are common in software data.

The fourth step is to identify data from atypical processes. These outliers are not representative of the normal operation of the process and should be removed before conclusions can be drawn about the attributes of a consistently performed process. Identification of atypical values is usually done using control charts, such as the XmR chart, although simpler outlier identification techniques can also be used, and techniques for identifying outliers specifically in the context of regression models are also available. Interquartile limits are used to identify outliers in Chapters 4 and 6. XmR charts are explored as a potentially superior alternative to interquartile limits for identifying outliers in Chapter 5. A variety of regression-based techniques for identifying influential outliers are used in Chapters 7 and 8.

Atypical production processes can be identified using design and coding times. Atypical modules can be identified using module size and defect density from reviews. Atypical reviews can be identified by using preparation rate and meeting review rate. (For PSP data, preparation rate is not applicable.)

The fifth step is to remove data from out-of-specification processes. The inspection rules can be considered equivalent to setting specification limits on the review (or inspection) process. Note that if the control limits are outside the specification limits, the process is not capable, and a distinction can be made between stable and conformant processes. For the PSP data, some inspection rules, such as the team size recommendations, are not pertinent.

The sixth step is to investigate the distributional assumptions for the defect data. Statistical techniques frequently make assumptions about the distribution of the data, with the normal distribution being the most common assumption. When averages are being tested, this is a reasonable assumption since the Central Limit Theorem indicates that averages approach a normal distribution as more observations are averaged. For software processes, however, individual observations must frequently be analyzed. Statistical distributions to be considered include the lognormal and Poisson; other distributions will be considered as appropriate. This is described in Chapter 6.

The seventh step is to model software quality in terms of the appropriate explanatory factors. Atypical data, outliers, and influential cases should be appropriately addressed. This may be by removing the atypical data or by considering the data sets with, and without, the atypical data as alternatives for the model. This is described in Chapter 7.

The eighth step is to extend the research beyond the context of the PSP classroom data, repeating the steps in the PSP analysis as appropriate. This analysis uses TSP and CMM high

maturity project data. Using project data allows the investigation of team effects. This is described in Chapter 8.

The final step is to summarize the results of each analysis with respect to each of the research questions. This is described in Chapter 9.

3.4 CONCERNS WITH GENERALIZING PSP-BASED ANALYSES

The primary source of data for most of these analyses is PSP. PSP is thoroughly instrumented, and there is a great deal of available data for analysis. Split data sets may be created across a number of different variables, such as *programming language* or *assignment*, as well as the PSPa data set, to verify results.

There are several concerns in using the PSP data, which can be summarized as limitations on the generalizability of analyses based on classroom data. Classroom data cannot be assumed to extrapolate to industrial settings. Subsequent analyses of TSP and CMM high maturity project data should mitigate these concerns but do not remove them since the project data available does not permit the same depth of analysis as the PSP data. Specific concerns include:

- **Classroom measures.** Potential explanatory variables that are eliminated from consideration by virtue of the data being classroom data may be significant in an industrial environment. Examples include variables associated with teamwork or larger systems. This concern is mitigated by analyzing TSP and high maturity project data, as described in Chapter 8, although further research in an industry setting is desirable.
- **Disciplined professionals.** The people who take the PSP course, or who choose to work in high maturity organizations, may not be typical of most software

professionals. A corollary is that people who finish the PSP course may differ from those who begin the class but do not finish. Students have generally been shown to provide an adequate model of software professionals within the scope of classroom-sized programs [Biffl and Gutjahr 2002, 269; Porter and Votta 1998]. This concern is mitigated by the analyses in Section 4.6.2, which suggest that professionals finishing the PSP course do not differ significantly from those who do not finish.

- **Small programs.** The systems built in real-world software projects are much larger than PSP assignments. For industrial projects, a design module may correspond to several code modules, since design is a higher level of abstraction than code. Similarly, a code module may consist of several procedures or subroutines, which may be independently reviewed. For PSP, however, data collection is at the level of the assignment rather than the module.
- **Programmer differences.** The surrogates, e.g., *years of programming experience*, available for testing the competence of programmers, while frequently used, are known to be inadequate [Curtis 1981]. This concern is mitigated by empirically measuring programmer ability using the data from the early PSP assignments, but using process and defect data to characterize the ability of programmers in an industry setting can lead to dysfunctional behavior, such as data falsification or inappropriate emphases, e.g., maximizing "productivity" at the expense of defects [Austin 1996].
- **Missing data.** Some contextual information, such as *programming language* and *years of programming experience*, was not consistently recorded in the PSP repository, therefore fewer data points are available for analyzing potential

explanatory variables. The richness of the PSP data mitigates this concern since in most cases the data sets remain relatively large even without the missing data.

- **Instructor differences.** There could be differences between classes due to the ability of the instructors [Johnson and Disney 1999, 331]. This should not be significant since the classes were taught by SEI-authorized instructors who have been through the PSP instructor training, and classes are typically co-taught by two to three instructors. This concern is mitigated by the analyses in Section 4.6.3, which suggest that there are no significant differences between the PSP classes.
- **No requirements phase.** PSP data captures time spent, defects injected, and defects removed in each PSP phase (or activity): planning, design, design reviews, coding, code reviews, compiling, testing, and postmortem. In PSP there is no requirements elicitation or analysis phase since the assignments are relatively straightforward; misunderstandings of the requirements are captured in the design, code, and test activities.

3.5 REMOVING INVALID PSP DATA

Johnson and Disney identified a number of concerns for PSP data validity centered around the manual reporting of personal data by the students [Johnson and Disney 1998; Johnson and Disney 1999]. In spite of reviews by the instructor and exhortations to approach the course professionally, they found about 5% of the data to be defective. Some of the classes of data errors they identified, such as errors of calculation, are irrelevant to this study because none of the analyses performed by the PSP students are used in my research – only the basic measures reported. Entry errors are a concern since they occur in the data collection stage and are difficult

to identify and correct, but less than 10% of the errors identified by Johnson and Disney were entry errors.

For the PSPb data set, internal data consistency errors were identified for 2.8% of the reported data. The steps where known defects were injected and where they were found are reported by the students; while there may be unknown defects latent in a program, acceptance of the assignment by the instructors sets a reasonable context for the defect data. Data is available for 112 classes with 1,345 students providing data from 10,223 assignments where a program was successfully completed. Inconsistencies between the total number of defects injected and removed resulted in the removal of 264 observations. For 21 observations, more defects were found in the design review than had been injected by that point in the process. For four observations, more defects were found in the code review than had been injected by that point in the process. Data errors were identified in 289 of the 10,223 observations, leaving 9,934 for analysis in the PSPb data set.

For the PSPa data set, errors were identified for 4.4% of the reported data. Data is available for 23 classes with 298 students providing data from 2,360 assignments where a program was successfully completed. In addition to internal consistency errors, two of the classes did not follow the standard ten programming assignments, therefore 299 observations were removed, leaving 21 classes. Inconsistencies between the total number of defects injected and removed resulted in the removal of 86 observations. For four observations, more defects were found in the design review than had been injected by that point in the process. No observations were removed because of more defects being found in the code review than had been injected by that point in the process. Data errors were identified in 390 of the 2,360 observations, leaving 1,970 for analysis in the PSPa data set.

A similar data cleanup was performed for the Hayes and Over study, but since repeated measures ANOVA was the primary analytic tool, an additional requirement was added that data be reported for all of the first nine assignments (assignment 10 was not analyzed) [Hayes 1996], with the result that data from 181 of the 298 students was usable [Hayes and Over 1997, 51].

The PSP data provides a wealth of data that provide natural data splits for replication of analyses. The PSPb and PSPa data sets are an obvious partition since they allow comparison of results to the Hayes and Over report [Hayes and Over 1997] insofar as the objectives of the different analyses are comparable. Splitting the data by *programming language* used is another natural partition since that technology factor affects productivity [Jones 1995] even though the impact of language on quality is one of the issues I am investigating.

Table 4 lists the number of observations for each assignment in the PSPa and PSPb data sets, including splits for the programming languages C and C++. The most frequently used languages for PSP are C and C++. Since programming language affects effort [Jones 1995], even if the impact on quality as measured by defect density is minimal, separating the data sets by programming language is a conservative decision.

Table 4 Sample Sizes for PSP Data Sets

Assignment	2001	2001 C	2001 C++	1997	1997 C	1997 C++
1A	**1086**	185	97	**221**	68	13
2A	**1117**	184	102	**234**	73	16
3A	**1115**	190	99	**226**	71	15
4A	**1091**	192	99	**227**	77	16
5A	**1059**	189	97	**201**	72	15
6A	**1032**	192	97	**201**	72	15
7A	**966**	178	90	**176**	63	12
8A	**901**	163	91	**176**	66	15
9A	**830**	152	82	**163**	61	13
10A	**737**	133	66	**145**	57	11
Totals	**9934**	**1758**	**920**	**1970**	**677**	**140**

Although the (PSPa, C++) data sets are small for useful statistical results, they are comparable to those found in many software studies, where data sets frequently have fewer than 30 data points, and the other data sets have ample data for useful analyses. For analyses focusing on performance with a disciplined process, the later assignments are preferred; in the first three assignments, a visible learning curve effect is noticeable as students become accustomed to following a defined personal process [Wesslen 1999, 177].

The statistical packages used in these analyses are JMP Version 4 and SAS Version 8.2, both of which are built by the SAS Institute in Cary, North Carolina. JMP was used for the analyses in Chapters 4 and 5, SAS was used for the analyses in Chapters 6 and 7, and both JMP and SAS were used for the analyses in Chapter 8.

4.0 EXPLORING THE FACTORS AFFECTING SOFTWARE QUALITY

4.1 THE RESEARCH QUESTION: EXPLORING QUALITY DRIVERS

The research in this chapter focuses on an initial exploration of the factors that significantly affect software quality and specifically on the effect of process discipline on software quality. The context of this analysis is software quality for individual professionals. Process-based factors are the variables of direct concern since the emphasis of my research is on the impact of disciplined processes, but hundreds of factors can affect the performance of a process. The majority of these factors contribute random noise to performance, but some are likely to systematically drive performance.

Many different factors have been suggested as affecting software quality, ranging from surrogates for ability, such as *years of experience*, to surrogates for technology and tools, such as the *programming language* used. The term surrogate is used because direct measurement of complex constructs such as "ability" and "technology" is difficult, and relatively easy-to-collect measures may suffice. Data for many of these surrogates is available for the Personal Software Process (PSP) course, and data from the ten PSP assignments enables exploring the effect of process discipline, and variables characterizing competent professionals and techniques and tools set the context.

The abilities of the PSP students and the effectiveness of their tools underlie the production and quality control processes for software work products such as design and code. While the emphasis of the exploratory data analysis is on the process variables that affect

software quality, confounding variables that could confound the results need to be appropriately addressed.

4.2 POTENTIAL EXPLANATORY VARIABLES

Potential explanatory variables identified in prior research can be divided into categories related to the application domain, the technologies used, the software engineering processes followed, and the ability of the individuals doing the work. These correspond to categories commonly used in causal analysis with cause-and-effect diagrams: people, equipment, methods, materials, measurement, and environment [Brassard and Ritter 1994, 25]. The results of my research deal with quantitative management, directly addressing the measurement category. Quality issues related to incoming material, i.e., the customer requirements, are outside the scope of my research. Although requirements volatility is a significant quality concern in software projects, requirements volatility is not an issue for PSP.

Although the focus of my research is on consistently following good practices, *process discipline* is only one of the factors that affect productivity and quality. Process and non-process contributors to software quality can be separated, but relationships between variables can be expected, e.g., capable programmers tend to consistently use effective techniques and tools.

The potential explanatory variables for software quality are identified based on the drivers in COQUALMO and empirical observations about software defects, as discussed in Section 2.3, and the available data from the PSP course. Some factors change systematically across PSP assignments as part of the teaching process, but many of the possible explanatory variables can be assumed to be constant within a PSP class (or within a project or team) and can therefore be ignored in this analysis.

The following factors from COQUALMO [Boehm et al. 2000, 254-268; Devnani-Chulani 1999] are not expected to change significantly within any of the data sets used in my research: *applications experience*, *architecture and risk resolution*, *application domain complexity*, *customer information*, *data base size*, *documentation match to life cycle needs*, *execution time constraint*, *language and tool experience*, *main storage constraint*, *multi-site development*, *personnel continuity*, *platform experience*, *platform volatility*, *precedentedness*, *product complexity*, *required development schedule*, *required reusability*, and *required software reliability*. Significant shifts in any of these factors should be proactively managed by a software project but are outside the scope of this analysis. Factors related to experience could affect the PSP learning curve, but should not be significant for the application and tools factors beyond the first few assignments; experience in using techniques such as reviews and design templates may be an issue as the PSP processes evolve.

The out-of-scope factors for PSP highlight the challenges in generalizing PSP results to software work in general. While it is reasonable to expect that significant factors in PSP will remain significant in an industrial context, other factors may need to be considered in building a comprehensive defect prediction model for industry projects. The factors that will be considered in this analysis address technology, process, and people issues. *Team composition* is not a consideration when analyzing the PSP data, but the increase in performance and decrease in variation associated with effective teams will be analyzed in the context of TSP and CMM high maturity project data in Chapter 8.

For the application domain factors, problem and solution complexity are indicated by lines of code (*LOC*) [Akiyama 1972; Lipow 1982; Gaffney 1984; Criscione, Ferree, and Porter 2001], which are available for all PSP assignments for new and changed code. The amount of

reused code is not available, therefore the size measure does not fully cover the functionality of the assignments.

Problem complexity would not appear to be a significant factor in PSP, given the relative simplicity of the assignments. Programs smaller than 10 KLOC are usually considered simple programs, between 10 and 100 KLOC are considered of medium complexity, and programs larger than 100 KLOC are considered relatively complex. Models frequently shift, even to different formulations, around these boundaries [Akiyama et al. 2002; Yang and Paradi 2004].

Solution complexity, or *product complexity*, is not constrained, and can be a significant source of variation [Weinberg 1998, 126-132; Criscione, Ferree, and Porter 2001; Takahashi and Kamayachi 1985]. A student's preferred style in optimizing memory space, speed, reliability (e.g., exception handlers), generality, reuse, etc., can lead to radically different solution complexities. Since PSP does not impose performance requirements, students have significant latitude in how they solve the problems – a latitude available in many industry contexts as well. Solution complexity can be indirectly measured by program size. The use of defect density should normalize out differences in solution complexity since *program size* can be used as a surrogate for solution complexity.

For technology factors, *programming language* captures an important technology consideration [Jones 1986; Lipow 1982]. For the small programs assigned to PSP students, tools should not be a differentiator, but the *programming language* could be a significant factor affecting productivity or quality.

For process factors, the two components are the production and review processes. Design templates for functional specifications, state specifications, logic specifications, and operational scenarios are introduced in PSP assignment 9. The use of more powerful design

techniques could result in an increase in design time. Design and code reviews are introduced in PSP assignment 7. Time spent in reviews and the number of defects removed in each review are captured. Defect removal effectiveness can be calculated based on the phase defects are injected in. More time in review should lead to higher defect removal effectiveness.

Process maturity is a generic concept that can be considered "low" for PSP0 and "high" for PSP3 [Boehm et al. 2000, 36]. One of the objectives of my research is to independently measure those aspects of the software process, e.g., review rates, that objectively define what "process maturity" is, i.e., the consistent execution of good engineering practices. It is interesting to note that in COQUALMO, *process maturity* has the highest impact of all factors on defect injection [Boehm et al. 2000, 260].

Because of the learning curve effects intrinsic to the PSP course, many of these analyses will focus on assignments 9 and 10. These two assignments incorporate design techniques and reviews that are considered good software engineering. In principle, these assignments represent a disciplined process employing recommended software techniques and tools. Assignments 9 and 10 can be separately analyzed to provide a split data set; to the degree significant learning curve effects remain in PSP between these two assignments, this may be desirable anyway.

Programmer/analyst capability is difficult to objectively determine. Factors such as *highest degree attained* and number of *years of programming experience* have been used as surrogates for programmer capability. Breadth of experience is usually considered a superior indicator of competence, and the *number of languages known* by the programmer is a plausible surrogate [Curtis, Krasner, and Iscoe 1988]. *Percent of time programming* in the previous year may indicate the steepness of the learning curve for students for the early assignments, and correlate with *applications experience* and *language and tool experience*. *Software developer*

experience with techniques may be an issue as the PSP design techniques are introduced since there may be learning curve effects even in the later assignments.

Ability, however, may be independent of any credentials an individual may possess. Programmer ability can be empirically identified for PSP assignments if comparative performance remains stable over assignments, i.e., top performers remain top performers as process discipline and techniques and tools are added. For industrial projects, programmer ability can be directly measured via the personnel review system [Banker, Datar, and Kemerer 1987; Banker, Datar, and Kemerer 1991], but the availability of such data may be constrained by privacy and confidentiality requirements. Untangling the effect of experience in general and experience with specific techniques and tools can be challenging, but the PSP data provides a direct and objective measure of programmer capability, which does not suffer from being either a surrogate or subjective.

4.3 DEFINING SOFTWARE QUALITY FOR PSP

Quality is a complex topic, with many different aspects [Garvin 1987]. Many definitions focus on customer perceptions, but for my research, the focus is on meeting stated customer requirements: the PSP assignments. In analyzing software quality for PSP, *defect density in testing* will be the surrogate measure for quality. Quality as perceived in terms of reliability (or other quality attributes) by the customer may differ from that observed in testing in terms of defects against the stated requirements. Conformance to requirements is a minimal definition of quality, but when PSP or TSP are applied to industry projects, very few to no defects, or the associated failures, are typically reported in the field [Ferguson et al. 1997; Webb and Humphrey 1999, 8], indicating that few defects escape the PSP and TSP processes.

The number of *defects removed in testing* is a plausible alternative. Defect density is the number of defects divided by size. Even for the relatively simple PSP assignments, *program size* ranges from 25 to over 300 LOC as shown in Figure 2.

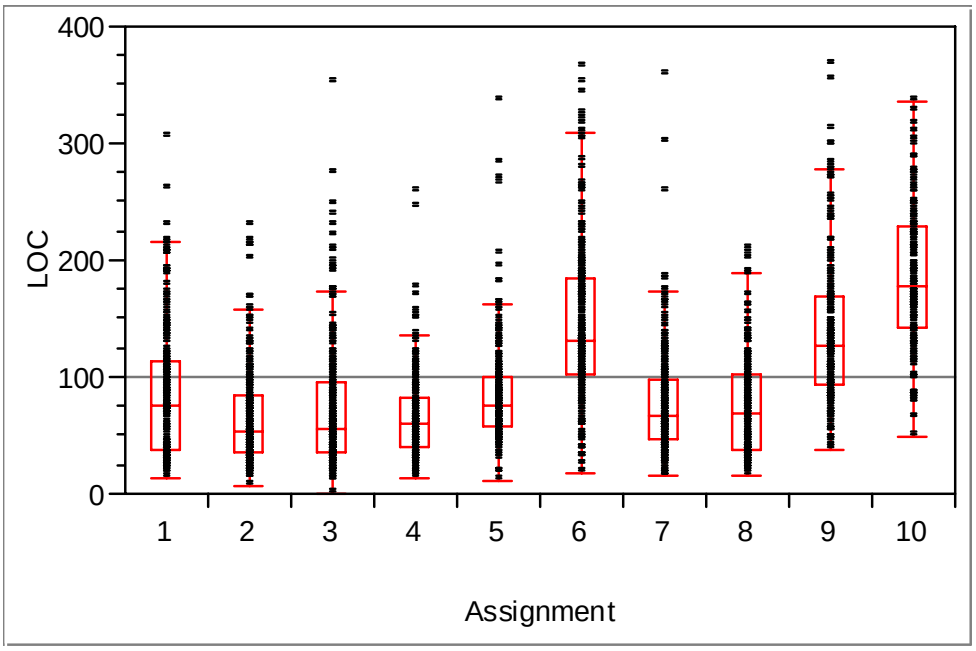

Figure 2 *Program Size* Across (PSPb, C)

The area of opportunity for injecting defects as measured by *program size* may vary an order of magnitude. Comparing the results for defect density and number of defects in the following analyses led to no additional insights or significant differences in the results (see Section 4.7.2). Defect density is a commonly used quality surrogate in the software industry [Gill and Kemerer 1991; Withrow 1990], and *defect density in testing* is the primary dependent variable and quality surrogate in these analyses. Unless otherwise noted, *defect density in testing* is the response variable in the tables and figures in this dissertation.

Defects removed in compile could be grouped with those removed in testing since the compiler is being used to detect syntactic defects that escaped from coding. The average effort per defect removed in testing is more than five times greater than that per defect removed in

compile (23.3 minutes/defect in testing versus 4.3 minutes/defect in compile for the PSPb data set). Focusing on *defect density in testing* therefore appears appropriate since it is relatively inexpensive to repair compile-time defects.

Direct measures of the effectiveness of some specific processes, and the quality of their work products, are available. An example for the design process is the defect density at the end of the production effort, e.g., the number of defects known to be present at the end of design per KLOC. For reviews, an appropriate quality surrogate is *defect removal effectiveness*, i.e., the percentage of defects detected of those known to be present at the beginning of the review. These are direct measures of the quality resulting from an interim process that contribute to the general quality surrogate, *defect density in testing*.

4.4 AN OVERVIEW OF SOME BASIC STATISTICS

In statistical tests, a null hypothesis H_0, such as the means of different treatments are equal or a regression coefficient is zero, is tested [Montgomery and Runger 1999, 296-304]. The objective of the researcher is usually to reject the null hypothesis, since the conclusion of interest is that the alternative hypothesis H_a is true. For example, H_a is that the mean for treatment A is greater than the mean for treatment B when the null hypothesis H_0, states they are equal (the treatment has no effect).

There is a possibility in any statistical test that the null hypothesis will be rejected when it is true, which is called a *Type I error*. The acceptable probability of a Type I error, $P(\text{reject } H_0 \text{ when } H_0 \text{ is true})$, is the significance level of the test and is denoted α, and $\alpha=0.05$ for these analyses. The *p-value* of a test is the estimated probability of a Type I error for a given data set under the relevant assumptions, typically normality and independence of the data. When *p-*

value<α, the null hypothesis, e.g., the means for two different treatments are equal, can be rejected. When the *p-value*>α, the proper conclusion is that the null hypothesis cannot be rejected.

There is also a possibility that the null hypothesis will be accepted when it is false, which is a *Type II error*. The probability of a Type II error, *P(failing to reject H_0 when H_0 is false),* is β. The *power* of a statistical test is (1-β). The sample size necessary to achieve a desired β may be calculated, but the power of a test is usually estimated rather than being specified in advance because of the practical difficulties frequently present in obtaining a large enough sample size to attain a desired power.

Nominal (or categorical) variables, such as *programming language*, can be analyzed using analysis of variance (ANOVA) techniques [Neter et al. 1996, 663-709]. Different treatments for an explanatory variable, e.g., different programming languages, are likely to have different variances, in which case the normal ANOVA assumption of equal variances for different treatments is not satisfied. The Welch test for means, which allows the variances to be unequal, is used rather than the standard ANOVA F test in such cases [Milliken and Johnson 1992, 27-28; SAS Institute 2000, 110]. For the Welch test, the means are weighted by the reciprocal of the sample variances of the treatment means. The Levene test is used to test the constancy of error variance. It is robust to departures from normality, and sample sizes need not be equal [Neter et al. 1996, 763].

For multiple comparisons, the *Each Pair, Student's t* and the *All Pairs, Tukey-Kramer honest significant difference* tests provide liberal and conservative comparison tests respectively [Milliken and Johnson 1992, 36-37; SAS Institute 2000, 100]. The *Each Pair* test is sized for individual comparisons and computes individual pair-wise comparisons, but there is no

protection across inferences, and the Type I error rate across the hypothesis tests is higher than that for individual differences. The *All Pairs* test is sized for all differences among the means and is a conservative alpha-level test if the sample sizes are different. Comparison circles graphically show the results of these comparison tests: the greater the overlap, the less likely there is a significant difference.

Regression models relate a dependent variable to one or more predictor variables. The coefficient of determination, R^2, can be used to judge the adequacy of the regression models [Montgomer and Runger 1999, 464-465]. Loosely speaking, R^2 describes the amount of variation in the data accounted for by the model. This coefficient may be adjusted when comparing models with differing numbers of predictor variables using the formula:

$$R^2_a = 1 - (\frac{n-1}{n-p})(1-R^2)$$

where there are n data points and p-1 predictor variables [Neter et al. 1996, 230-231].

A statistical correlation between two variables does not necessarily indicate a cause-and-effect relationship. Although we may say that there are statistically significant results due to variable X for variable Y, the causal relationship should be based on a conceptual model of how X drives Y that the data supports. Even when the analysis supports the theory, it is always possible that unidentified factors are confounding the analysis or that correlations between variables are masking the effects of each variable considered individually.

4.5 CONFIRMING PSP QUALITY TRENDS

As can be observed in Figure 3, the software quality trend across the PSP major processes is apparent, confirming prior analyses [Hayes and Over 1997; Wesslen 2000]. Performance

improves and variation decreases as the PSP processes become more sophisticated: PSP0 is the baseline process, PSP1 adds size estimating and test reports, PSP2 inserts reviews and design templates, and PSP3 introduces cyclic development. The figure shows the differences in *defect density in testing* for each *PSP major process* for the PSPb data set.

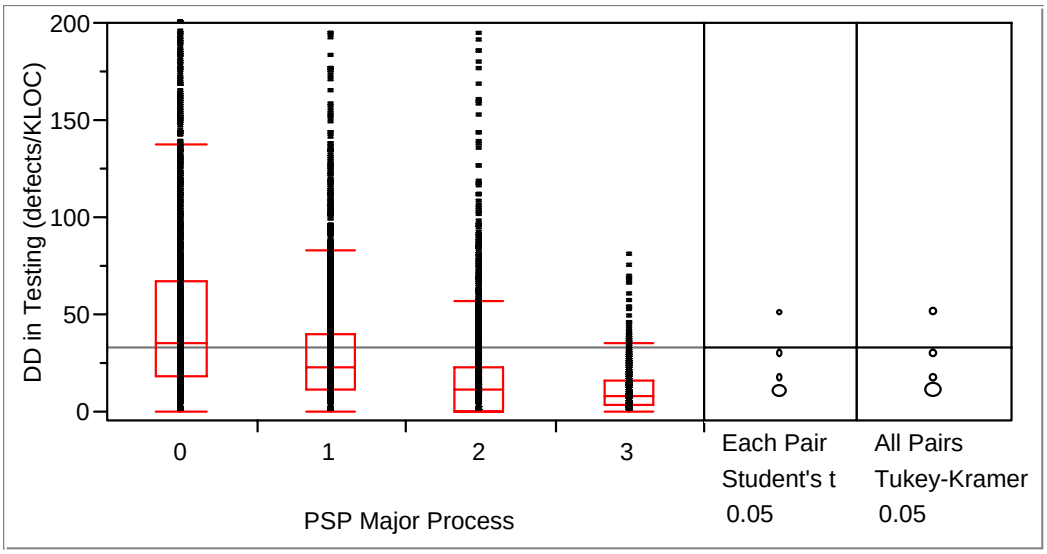

Figure 3 Trends in Software Quality

The *Each Pair* and *All Pairs* tests indicate that the means for each of the *PSP major processes* are significantly different, as illustrated by the comparison circles in the figure.

The ANOVA results for the effect of the *PSP major process* on *defect density in testing* are shown in Table 5. The null hypothesis is $H_0 : \mu_{PSP0} = \mu_{PSP1} = \mu_{PSP2} = \mu_{PSP3}$ with alternative hypothesis H_a*: not all of the means are equal.* The table includes the ANOVA information for the two data sets used to analyze the *PSP major process*, including the information for the treatment levels (the four processes), the error term, and the total model. The information includes degrees of freedom (DF), sum of squares (SS), mean square (MS), the F Ratio and *p-value* for the F test, and the adjusted coefficient of determination (R^2_a) for the model.

Table 5 ANOVA for *PSP Major Process*

Source		**PSPa**	**PSPb**
Model	DF	3	3
	SS	330630	2122446
	MS	110210	707482
Error	DF	1966	9930
	SS	3317337	16072027
	MS	1687	1619
Total	DF	1969	9933
	SS	3647966	18194474
F Ratio		102.8^{W}	532.7^{W}
Prob > F		$<.0001^{W}$	$<.0001^{W}$
R^2_a		0.0892	0.1164

The effect of the *PSP major process* on *defect density in testing* was shown to be statistically significant for both of the data sets. This indicates that *PSP major process* is a useful predictor variable for *defect density in testing*, at least within the context of the PSP class. As might be expected, and as demonstrated by the Levene test for equal variances, variation decreases as more sophisticated processes are used. The Welch test is therefore preferred over the F test, which assumes equal variances, for testing the differences in mean performance and is noted with the symbol ^{W} next to the F ratio.

The estimates of the means for *defect density in testing* at the four levels of *PSP major process*, and the associated standard errors for the means, are listed in Table 6 for the data sets. A model can be built using indicator variables for estimating the defect density associated with each level of the *PSP major process*. The formula could be written as:

(*Defect density in testing*) = (*PSP0 Estimate*) (*PSP0 Indicator*)

+ (*PSP1 Estimate*) (*PSP1 Indicator*)

+ (*PSP2 Estimate*) (*PSP2 Indicator*)

+ (*PSP3 Estimate*) (*PSP3 Indicator*)

where the indicator variable is either 0 or 1 depending on which level of the *PSP major process* is applicable. A more concise notation is:

$$(\textit{Defect density in testing}) = \beta_{\textit{PSP Major Process}} X_{\textit{PSP Major Process}}$$

where β_j is an estimate of the mean for the j^{th} level of the categorical variable (*PSP major process* in this instance) and X_j is the corresponding indicator variable. For models with multiple categorical variables, the i^{th} categorical variable with j levels is described by $\beta_{i,j} X_{i,j}$.

For the null hypothesis $H_0 : \mu_{PSP0} = \mu_{PSP1} = \mu_{PSP2} = \mu_{PSP3}$ in the following tables of level estimates, the p-values are captured in the column header for the model. A *p-value*<0.05 is indicated with *, a *p-value*<0.01 is indicated with **, a *p-value*<0.001 is indicated with ***, and a *p-value*<0.0001 is indicated with ****.

Table 6 Estimates for *PSP Major Process* Levels

Level	PSPa**** (std err)	PSPb**** (std err)
PSP0	45.6 (2.2)	51.6 (1.0)
PSP1	28.5 (1.4)	30.3 (0.6)
PSP2	16.2 (0.9)	18.0 (0.5)
PSP3	8.7 (0.9)	11.4 (0.5)

The differences between the *PSP major processes* are both statistically and practically significant. The differences of 5.2 to 1 for PSPa and 4.5 to 1 for PSPb show a decrease in *defect density in testing* from 78-81%. This is a quality improvement that would be of interest and value to most software professionals.

Some statistical assumptions are violated in this analysis. First, the number of data points for each *PSP major process* is different since some students drop out of the PSP course before finishing, leading to unbalanced data sets with missing data. Second, because each data set contains multiple data points per student, the usual assumption of independence is violated. Third, it cannot be assumed that any software data set is normally distributed; software data is usually skewed. These concerns are systematically addressed in the multiple regression models and mixed models in Chapter 7, but they are ignored in the exploratory data analysis in this chapter since the techniques used are robust.

This analysis also ignores the possible influence of outliers. Excluding outliers without causal analysis, as well as including them when they are atypical, can skew results. Statisticians

debate whether it is preferable to remove outliers before doing statistical analysis [Orr, Sackett, and Dubois 1991; Judd and McClelland 1989, 207-237]. The remaining analyses in this chapter are performed both with and without outliers, using the interquartile limits (IQL) technique to identify outliers. Chapter 5 compares the IQL technique to XmR charts as an outlier identification technique.

4.6 EXPLORING THE POTENTIALLY CONFOUNDING VARIABLES

Before exploring the primary variables for process discipline and individual differences, it is desirable to investigate a number of confounding variables that could confound the analyses if not appropriately addressed. The hypotheses for confounding variables will be considered statistically significant for $\alpha=0.05$.

Confounding variables related to people include those associated with learning curve effects, programmer differences, instructor differences, credentials, experience, recent experience, and breadth of experience. In some cases these potentially confounding variables are specific to the classroom environment, e.g., instructor differences. In other cases, such as *years of experience*, some studies have found the variable significantly related to software quality, as described in Chapter 2, although the results of other studies may have differed.

Hypothesis C1 is that there is a difference in *defect density in testing* between assignments 9 and 10. Since the process, tools, and application domain are essentially the same, a significant difference indicates a learning curve effect is occurring.

Hypothesis C2 is that there is a difference in *defect density in testing* between the students who finish all ten PSP assignments and those who do not. A significant difference would suggest a selection effect that would affect the generalizability of the results.

Hypothesis C3 is that there is a difference in *defect density in testing* between the PSP classes (the different offerings of the PSP course). A significant difference would suggest that classes are not similar learning environments, perhaps due to different instructors, changes in the teaching materials, or changes in the student population.

Hypothesis C4 is that the *highest degree attained* is related to *defect density in testing*. A significant difference would indicate that educational credentials affect programming performance and could indicate that credentials are a reasonable surrogate for programmer ability.

Hypothesis C5 is that *years of experience* is related to *defect density in testing*. A significant difference would indicate that experience affects programming performance and could indicate that experience is a reasonable surrogate for programmer ability.

Hypothesis C6 is that *percent of time programming* in the previous year is related to *defect density in testing*. A significant difference would indicate that recent experience affects programming performance and could indicate that recent experience is a reasonable surrogate for programmer ability. This should not be a factor by the end of the class but could affect performance on early assignments.

Hypothesis C7 is that the *number of programming languages known* is related to *defect density in testing*. A significant difference would indicate that that the *number of programming languages known*, which presumably corresponds to a diversity of programming experiences, is a reasonable surrogate for breadth of experience, which is an aspect of programmer ability.

Hypothesis C8 is that the *programming language* used is related to *defect density in testing*. A significant difference would indicate that that technology used is affecting software quality. If no significant difference is found, technology may still affect other concerns, such as productivity.

Since differences between individuals can contribute over half of the variance in performance [Curtis 1981], the values of the (adjusted) coefficient of determination, R^2_a, are likely to be small for any specific explanatory variable, and a statistically significant explanatory variable with $R^2_a > 0.05$ may be a useful addition to a defect prediction model. Programming is a creative process and a social activity [Cockburn 2004, 7-10]. In the social sciences, $R^2_a = 0.30$ might be considered acceptable [SAS Institute 1989, 15], and the multiple regression models described in Chapter 7 should exceed that value (and larger values of R^2_a are desirable). The focus of this analysis is identifying which explanatory variables are worthy of further consideration and exploration. Later analyses will consider data transformations and correlations that may affect the relationships.

In this section, the data sets used are primarily the PSPa and PSPb data sets since the analysis of potential confounding variables, such as the *programming language* used, is intended to help identify useful criteria for splitting the data in later analyses. A secondary criterion is assignment since that removes process and problem complexity differences that might confound the analyses. Where it is appropriate to focus on data from relatively mature processes, the data sets are split by assignments 9 and 10. For the comparison of programmers who finished all ten assignments versus those who did not, in Section 4.6.2, the appropriate data split is by assignments 1 and 2.

4.6.1 Assignment (9A Versus 10A)

As reported in various studies of PSP, there are consistent, statistically significant changes in *defect density in testing* across assignments. Even when the focus of these analyses is on assignments 9 and 10 to minimize the effects of process changes and learning, if learning curve effects between these two assignments are significant, then the data may need to be split by assignment. Figure 4 illustrates the differences in *defect density in testing* between assignments 9 and 10 for the PSPb data set, including outliers.

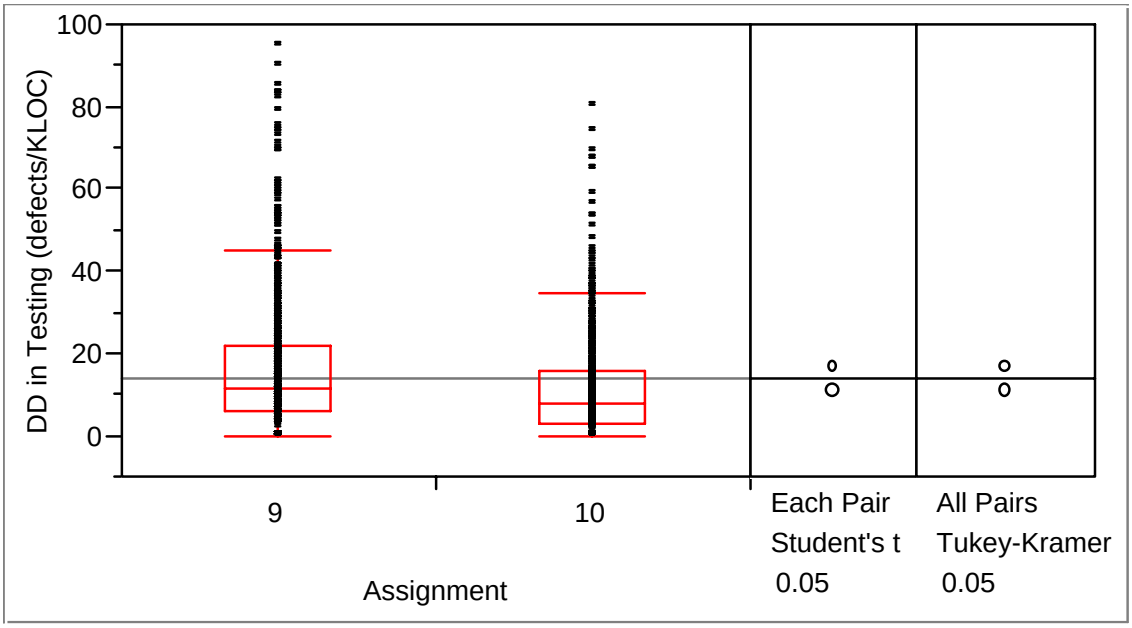

Figure 4 Differences Between Assignment 9 and Assignment 10

The *Each Pair* and *All Pairs* tests indicate that the means for assignments 9 and 10 are significantly different, as illustrated by the comparison circles in the figure.

The ANOVA results for the effect of the *assignment* on *defect density in testing* are shown in Table 7. The null hypothesis is $H_0 : \mu_{9A} = \mu_{10A}$ with alternative hypothesis $H_a : \mu_{9A} \neq \mu_{10A}$.

Table 7 ANOVA for *Assignment* (9A vs 10)

Source		PSPa	PSPb
Model	DF	1	1
	SS	1851.8	12062.4
	MS	1851.8	12062.4
Error	DF	306	1565
	SS	63791.1	480130.0
	MS	208.5	306.8
Total	DF	307	1566
	SS	65642.9	492192.4
F Ratio		9.3^{W}	40.9^{W}
Prob > F		0.0025^{W}	$<.0001^{W}$
R^2_a		0.0250	0.0239

The effect of *assignment* on *defect density in testing* was shown to be statistically significant for both of the data sets. This indicates that *assignment* is a useful predictor variable for *defect density in testing*, at least within the context of the PSP class.

The estimates of the means for *defect density in testing* at the different levels of *assignment*, and the associated standard errors for the means, are listed in Table 8 for the data sets including outliers. The model can be expressed as:

$$(\textit{Defect density in testing}) = \beta_{Assignment} X_{Assignment}$$

where $\beta_{Assignment}$ is the level for the *assignment* (9A or 10A) and $X_{Assignment}$ is an indicator variable for whether that *assignment* is the correct one for the observation.

Table 8 Estimates for *Assignment* Levels

Levels	PSPa** (std err)	PSPb**** (std err)
9A	13.65 (1.31)	16.95 (0.69)
10A	8.74 (0.94)	11.39 (0.53)

The ANOVA results for the data sets excluding outliers are provided in Table 9. Outliers were identified with respect to *defect density in testing*.

Table 9 ANOVA for *Assignment* (9A vs 10A) Excluding Outliers

Source		PSPa	PSPb
Model	DF	1	1
	SS	1169.2	5971.8
	MS	1169.2	5971.8
Error	DF	298	1486
	SS	28319.1	140680.2
	MS	95.0	94.7
Total	DF	299	1487
	SS	29488.3	146652.0
F Ratio		12.6^{W}	64.7^{W}
Prob > F		0.0005^{W}	$<.0001^{W}$
R^2_a		0.0364	0.0401

The ANOVA results for the data sets excluding outliers are similar to those for the data sets with outliers. This indicates that *assignment* is a useful predictor variable for *defect density in testing*.

The estimates of the means for *defect density in testing* at the different levels of *assignment*, and the associated standard errors for the means, are listed in Table 10 for the data sets excluding outliers.

Table 10 Estimates for *Assignment* Levels Excluding Outliers

Levels	PSPa*** (std err)	PSPb**** (std err)
9A	11.66 (0.84)	13.40 (0.38)
10A	7.70 (0.73)	9.39 (0.32)

The ANOVA results indicate there is a consistent, statistically significant difference between assignments 9 and 10 for *defect density in testing*. The data sets will be split by assignment where appropriate in subsequent analyses.

The difference between assignments 9 and 10 could be due to the process changes between PSP2.1 and PSP3 involved in adopting cyclic development. Conceptually, this difference seems minor, but the size of the programs in assignment 10 is noticeably greater than those in the earlier assignments (see Figure 2). There may be learning curve effects due to new software engineering techniques, since PSP2.1 introduces design techniques for assignment 9. This may be a significant process change for those who have not used these design techniques before, even if they have used other design techniques on earlier PSP assignments. It seems unlikely that there are significant learning curve effects due to the application domain.

It is unclear whether the difference between assignments should be attributed to process differences or learning curve effects (and the two are related since there are learning curve effects associated with adopting new processes). As observed in previous PSP studies [Ferguson

et al. 1997], performance continues to improve after the PSP class, so the presence of on-going learning curve effects would not be surprising.

4.6.2 Finishing All Ten Assignments Versus Not Finishing

The data for each of the PSP0 assignments can be examined to determine whether those who finish all ten assignments differ from those who begin the course but drop out for some reason. For the PSPa data set, 103 students finished all ten assignments; for PSPb, 573 students finished all ten. Figure 5 illustrates the differences in *defect density in testing* for the students finishing all ten assignments versus those not finishing the course for (PSPb, 1A, Outliers).

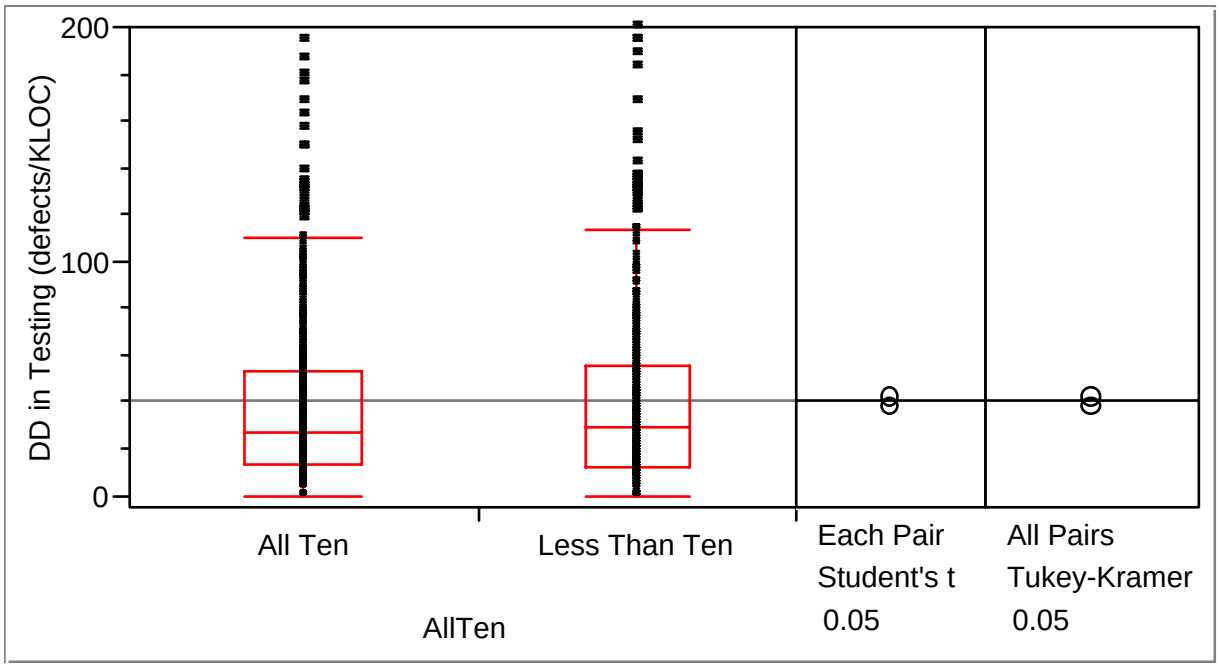

Figure 5 Differences Between *All Ten* Assignments Versus *Less Than Ten*

The *Each Pair* and *All Pairs* tests indicate that the means for the students finishing all ten assignments are not significantly different from those not finishing the course, as illustrated by the comparison circles in the figure.

The ANOVA results for the effect of finishing *all ten* assignments on *defect density in testing* are shown in Table 11. The null hypothesis is $H_0 : \mu_{AllTen} = \mu_{LessThanTen}$ with alternative hypothesis $H_a : \mu_{AllTen} \neq \mu_{LessThanTen}$.

Table 11 ANOVA for Finishing *All Ten* Assignments

Source		**PSPa 1A**	**PSPa 2A**	**PSPb 1A**	**PSPb 2A**
Model	DF	1	1	1	1
	SS	7729.5	3332.1	3826.3	3822.2
	MS	7729.5	3332.1	3826.3	3822.2
Error	DF	219	232	1084	1115
	SS	435605.5	939809.9	2017638.5	4011508.5
	MS	1989.1	4050.9	1861.3	3597.8
Total	DF	220	233	1085	1116
	SS	443335.0	943142.0	2021464.8	4015330.7
F Ratio		4.2[W]	0.8	2.0[W]	1.1[W]
Prob > F		0.0423[W]	0.3654	0.1574[W]	0.3053[W]
R^2_a		0.0129	-0.0008	0.0010	0.0001

The effect of finishing *all ten* assignments on *defect density in testing* was shown to be statistically significant for only one of the four data sets: (PSPa, 1A, Outliers). This indicates that finishing *all ten* assignments is unlikely to be a useful predictor variable for *defect density in testing*.

The estimates of the means for *defect density in testing* for those finishing *all ten* assignments versus those finishing *less than ten*, and the associated standard errors for the means, are listed in Table 12 for the data sets including outliers. The model can be expressed as:

(*Defect density in testing*) = $\beta_{All\ Ten} X_{All\ Ten}$

where $\beta_{All\ Ten}$ is the level for (not) finishing *all ten* assignments and $X_{All\ Ten}$ is an indicator variable for whether *all ten* or *less than ten* is the correct one for the observation.

Table 12 Estimates for Levels of Finishing *All Ten* Assignments

Levels	PSPa 1A* (std err)	PSPa 2A (std err)	PSPb 1A (std err)	PSPb 2A (std err)
All Ten	32.3 (3.0)	51.9 (5.2)	39.0 (1.6)	51.6 (2.3)
Less Than Ten	44.1 (5.0)	44.3 (6.2)	42.8 (2.1)	55.3 (2.8)

The ANOVA results for the data sets excluding outliers are provided in Table 13. Outliers were identified with respect to *defect density in testing*.

Table 13 ANOVA for Finishing *All Ten* Assignments Excluding Outliers

Source		PSPa 1A	PSPa 2A	PSPb 1A	PSPb 2A
Model	DF	1	1	1	1
	SS	1208.0	2382.9	21.6	15.8
	MS	1208.0	2382.9	21.6	15.8
Error	DF	202	215	1021	1039
	SS	109112.2	139693.0	737581.9	1085639.0
	MS	540.2	649.7	722.4	1044.9
Total	DF	203	216	1022	1040
	SS	110320.2	142075.9	737603.5	1085654.7
F Ratio		2.2	3.7	0.03	0.02
Prob > F		0.1364	0.0568	0.8627	0.9023
R^2_a		0.0061	0.0122	-0.0010	-0.0010

The ANOVA results for the data sets excluding outliers are similar to those for the data sets with outliers, differing only for the one case that was shown to be statistically significant when outliers were included. The preponderance of the evidence therefore indicates that finishing *all ten* assignments is unlikely to be a useful predictor variable for *defect density in testing*.

The estimates of the means for *defect density in testing* for those finishing *all ten* assignments versus those finishing *less than ten*, and the associated standard errors for the means, are listed in Table 14 for the data sets excluding outliers.

Table 14 Estimates for Levels of Finishing *All Ten* Assignments Excluding Outliers

Levels	PSPa 1A (std err)	PSPa 2A (std err)	PSPb 1A (std err)	PSPb 2A (std err)
All Ten	27.1 (2.2)	37.7 (2.8)	33.2 (1.1)	41.5 (1.4)
Less Than Ten	31.9 (2.4)	31.0 (2.2)	32.9 (1.2)	41.2 (1.5)

The preponderance of the evidence in the ANOVA results indicates that finishing the course should not be considered a significant explanatory variable. Although a statistically significant difference was shown for *defect density in testing* in (PSPa, 1A, Outliers), the statistical significance is marginal and disappears when outliers are excluded. This suggests that people finishing the PSP course are reasonably typical of programmers who choose to take the PSP course.

Excluding observations with missing data, i.e., excluding observations for students who did not finish all ten assignments, is advantageous for some statistical analyses, such as repeated measures ANOVA. For example, the Hayes and Over analysis of the PSPa data only used the data for those students finishing assignments 1-9 (assignment 10 was not considered in their analysis) [Hayes and Over 1997, 51]. This result supports the generalizability of such analyses of PSP data.

4.6.3 PSP Classes

There are two situations that could cause systemic differences across PSP classes: 1) changes in the teaching materials used in the PSP class, or 2) differences between instructors. The possibility of a trend due to systemic changes in the student population does not appear likely, since there are no known reasons for a systemic change in the student population for PSP,

although sporadic cases where exceptionally poorly-prepared or well-prepared classes could occur. The PSP class has been based on the text A Discipline for Software Engineering since its publication in 1995 [Humphrey 1995]; prior classes used drafts of the manuscript. PSP instructors are all qualified and authorized by the Software Engineering Institute, and instruction is typically done by teams of instructors. There would not appear to be any reason for changes in PSP class performance over time, although there might be cases where particular classes might have significantly different results because of special causes of variation.

Viewing the *PSP classes* as a sequential variable over time allows an exploration of time-based shifts in quality. The regression results for the effect of *PSP class* on *defect density in testing* are shown in Table 15, where the *PSP class* is a sequence number rather than a nominal variable, which allows an exploration of systemic trends over time. For the regression model:

$$(\textit{Defect density in testing}) = \beta_0 + \beta_1 (\textit{PSP class})$$

the null hypothesis tested is $H_0: \beta_1 = 0$ with alternative hypothesis $H_a: \beta_1 \neq 0$.

Table 15 Regression Models for *PSP Class*

Source		PSPa 9A	PSPa 10A	PSPb 9A	PSPb 10A
Model	DF	1	1	1	1
	SS	188.9	5.6	1685.6	395.2
	MS	188.9	5.6	1685.6	395.2
Error	DF	161	143	828	735
	SS	45302.2	18294.4	328541.2	149508.1
	MS	281.4	127.9	396.8	203.4
Total	DF	162	144	829	736
	SS	45491.1	18300.0	330226.8	149903.2
F Ratio		0.7	0.04	4.2	1.9
Prob > F		0.4138	0.8339	0.0396	0.1638
R^2_a		-0.0020	-0.0067	0.0039	0.0013

The effect of *PSP class* on *defect density in testing* was shown to be statistically significant for only one of the four data sets: (PSPb, 9A, Outliers). This indicates that *design review class* is unlikely to be a useful predictor variable for *defect density in testing*.

The parameter estimates for the regression models for *PSP class* trends, and the associated standard errors, are listed in Table 16 for the data sets including outliers. For the null hypothesis $\beta_i = 0$ in the following tables of parameter estimates, a *p-value*<0.05 is indicated with *, a *p-value*<0.01 is indicated with **, a *p-value*<0.001 is indicated with ***, and a *p-value*<0.0001 is indicated with ****.

Table 16 Estimates for *PSP Class*

Parameter	PSPa 9A (std err)	PSPa 10A (std err)	PSPb 9A (std err)	PSPb 10A (std err)
β_0 (Intercept)	11.97**** (2.43)	9.05**** (1.78)	20.49**** (1.85)	13.23**** (1.42)
β_1	0.13 (0.15)	-0.02 (0.11)	-0.04* (0.02)	-0.02 (0.02)

Some classes may be atypical; the *PSP class* is naturally a nominal variable. The *Each Pair* and *All Pairs* tests for a nominal representation of the class data indicate that some PSP classes are atypical, as illustrated by the comparison circles in Figure 6 for (PSPb, 9A, Outliers).

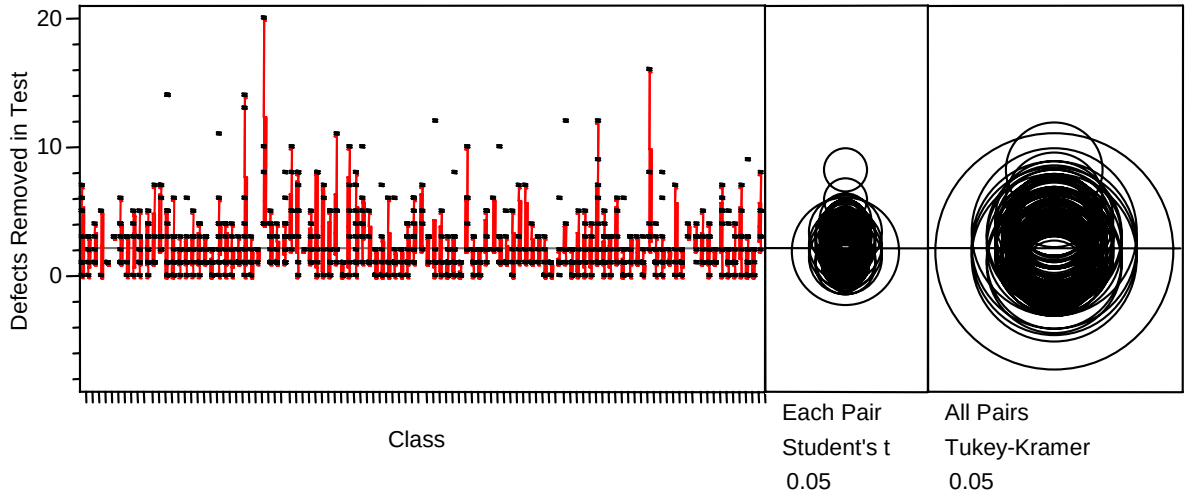

Figure 6 *PSP Classes* Over Time

The class with the largest average *defect density in testing* (44.0 defects/KLOC) appears clearly different from the rest of the classes, which range from 1.6 to 30.2 defects/KLOC. Upon further examination of this, and four other classes where *defect density in testing* was abnormally high at the end of the course, the anomalies appear related to a limited number of students having trouble with one or more assignments.

In the case of the class with an average defect density of 44.0 defects/KLOC, two students discovered more than ten defects in testing in assignment 10 (14 defects in 238 LOC and 12 defects in 163 LOC). Since only five students finished all ten programs in this class, the average and standard deviation for *defect density in testing* are high. Similarly for the other four atypical classes, there are one or two students with an unusually high number of defects. In one case, a student had 8 defects in a 35 LOC program on assignment 10, for a defect density of 229 defects/KLOC. Similarly for PSPa, one class was atypical because of one student with 11 defects in a 72 LOC program for a defect density of 153 defects/KLOC.

Combining a small number of finishing students with one or two students struggling to finish assignment 9 and/or 10 leads to the occasional atypical class, which is not unexpected given the large number (112) of classes being analyzed. Since there does not appear to be a statistically or practically significant trend over time, it seems reasonable to conclude that, in general, PSP classes are relatively stable learning environments, although a small number of students may have trouble on some assignments. Atypical student data will be removed as appropriate for later analyses rather than excluding class data.

To support this conclusion, additional analyses were run for the trends with outliers excluded. Outliers were identified with respect to *defect density in testing* for students rather than classes. The ANOVA results for the data sets excluding outliers are provided in Table 17.

Table 17 Regression Models for *PSP Class* Excluding Outliers

Source		PSPa 9A	PSPa 10A	PSPb 9A	PSPb 10A
Model	DF	1	1	1	1
	SS	1.6	88.9	209.2	10.0
	MS	1.6	88.9	209.2	10.0
Error	DF	156	136	788	702
	SS	17551.9	7130.2	98369.6	50231.6
	MS	112.5	52.4	124.8	71.6
Total	DF	157	137	789	703
	SS	17553.5	7219.1	98578.9	50241.6
F Ratio		0.01	1.7	1.7	0.1
Prob > F		0.9053	0.1950	0.1958	0.7083
R^2_a		-0.0063	0.0051	0.0009	-0.0012

None of the regression models shows a statistically significant regression for *PSP classes* for *defect density in testing* for the data sets excluding outliers.

The parameter estimates for the regression models for *PSP class*, and the associated standard errors, are listed in Table 18 for the data sets excluding outliers.

Table 18 Estimates for *PSP Class* Excluding Outliers

Parameter	PSPa 9A (std err)	PSPa 10A (std err)	PSPb 9A (std err)	PSPb 10A (std err)
β_0 **(Intercept)**	11.50**** (1.55)	8.12**** (1.15)	14.97**** (1.07)	9.69**** (0.87)
β_1	0.01 (0.09)	-0.09 (0.07)	-0.02 (0.01)	-0.004 (0.01)

The preponderance of the evidence in the ANOVA tables indicates that *PSP class* is unlikely to be a useful predictor variable. Although a statistically significant difference was shown for *defect density in testing* in (PSPb, 9A, Outliers), the statistical significance is marginal and disappears when outliers are excluded. This suggests that the *PSP class* is a reasonably stable learning environment and that excluding outliers adequately addresses any concerns related to differences between classes or instructors.

4.6.4 Highest Degree Attained

In analyzing the *highest degree attained*, a full representation of undergraduate and graduate degrees is available for PSPb: BE (Bachelor of Engineering), BS (Bachelor of Science), MS (Master of Science), and PhD (Philosophy Doctorate). For PSPa, there are only two people with BE or PhD degrees, therefore only the BS and MS are considered in those models. The *Each Pair* and *All Pairs* tests indicate that the means for students with different academic credentials (degrees) are not significantly different, as illustrated by the comparison circles in Figure 7 for (PSPb, 9A, Outliers).

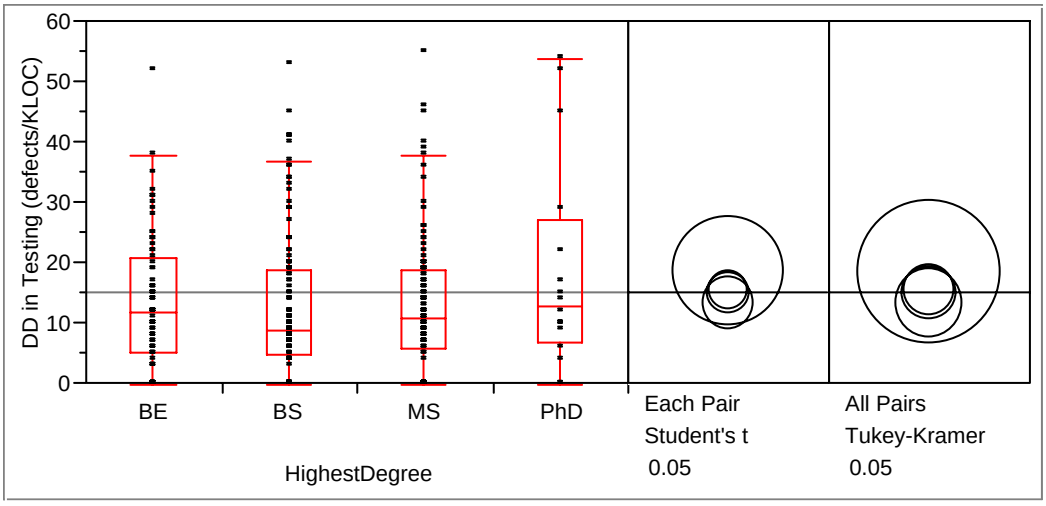

Figure 7 Differences for *Highest Degree Attained*

The ANOVA results for the effect of *highest degree attained* on *defect density in testing* are shown in Table 19. The null hypothesis is $H_0 : \mu_{BE} = \mu_{BS} = \mu_{MS} = \mu_{PhD}$ with alternative hypothesis H_a*: not all of the means are equal.*

Table 19 ANOVA for *Highest Degree Attained*

Source		PSPa 9A	PSPa 10A	PSPb 9A	PSPb 10A
Model	DF	1	1	3	3
	SS	40.7	470.4	437.8	327.2
	MS	40.7	470.4	145.9	109.1
Error	DF	85	78	321	269
	SS	14242.3	4976.4	107895.9	35963.3
	MS	167.6	63.8	336.1	133.7
Total	DF	86	79	324	272
	SS	14282.9	5446.8	108333.7	36290.6
F Ratio		0.2	4.3[W]	0.4	0.8
Prob > F		0.6236	0.0500[W]	0.7287	0.4861
R^2_a		-0.0089	0.0746	-0.0053	-0.0020

The effect of the *highest degree attained* on *defect density in testing* was shown not to be statistically significant for all of the data sets. This indicates that the *highest degree attained* is not a useful predictor variable for *defect density in testing*.

The estimates of the means for *defect density in testing* at the different levels of *highest degree attained*, and the associated standard errors for the means, are listed in Table 20 for the data sets including outliers. The model can be expressed as:

$$(\textit{Defect density in testing}) = \beta_{\textit{Highest Degree}} X_{\textit{Highest Degree}}$$

where $\beta_{\textit{Highest Degree}}$ is the level for the *highest degree attained* and $X_{\textit{Highest Degree}}$ is an indicator variable for whether that *highest degree attained* is the correct one for the observation.

Table 20 Estimates for *Highest Degree Attained* Levels

Levels	PSPa 9A (std err)	PSPa 10A (std err)	PSPb 9A (std err)	PSPb 10A (std err)
BE	---	---	13.49 (1.28)	10.90 (1.21)
BS	12.20 (1.67)	6.40 (0.83)	15.71 (2.07)	11.43 (1.31)
MS	13.78 (2.44)	12.00 (2.58)	15.13 (1.54)	9.95 (1.14)
PhD	---	---	18.69 (4.36)	6.00 (2.69)

The ANOVA results for the data sets excluding outliers are provided in Table 21. Outliers were identified with respect to *defect density in testing*.

Table 21 ANOVA for *Highest Degree Attained* Excluding Outliers

Source		PSPa 9A	PSPa 10A	PSPb 9A	PSPb 10A
Model	DF	1	1	3	3
	SS	165.5	153.2	166.7	174.2
	MS	165.5	153.2	55.6	58.1
Error	DF	83	75	299	261
	SS	8909.1	3128.0	26467.4	19871.9
	MS	107.3	41.7	88.5	76.1
Total	DF	84	76	302	264
	SS	9074.6	3281.2	26634.0	20046.1
F Ratio		1.5	2.4[W]	0.6	0.8
Prob > F		0.2179	0.1347[W]	0.5976	0.5159
R^2_a		0.0064	0.0340	-0.0037	-0.0027

The ANOVA results for the data sets excluding outliers are similar to those for the data sets with outliers. All of the analyses, including and excluding outliers, indicate that the *highest degree attained* is not a useful predictor variable for *defect density in testing*.

The estimates of the means for *defect density in testing* at the different levels of the *highest degree attained*, and the associated standard errors for the means, are listed in Table 22 for the data sets excluding outliers.

Table 22 Estimates for *Highest Degree Attained* Levels Excluding Outliers

Levels	PSPa 9A (std err)	PSPa 10A (std err)	PSPb 9A (std err)	PSPb 10A (std err)
BE	---	---	12.94 (1.18)	10.08 (1.10)
BS	10.59 (1.26)	6.00 (0.74)	11.11 (0.91)	9.60 (0.96)
MS	13.77 (2.44)	9.33 (2.01)	11.28 (0.86)	8.87 (0.82)
PhD	---	---	11.38 (2.30)	6.00 (2.69)

Although degree credentials are not a significant factor for PSP assignments, educational credentials may contribute to improved performance for more complex programs, where deeper, more extensive domain or engineering knowledge may be crucial to understanding both the problem and potential solutions.

4.6.5 Years of Programming Experience

Programmer skill is frequently identified as a major contributor to quality, and *years of experience* is frequently used as a surrogate for skill. The regression results for the effect of *years of experience* on *defect density in testing* are shown in Table 23. For the regression model:

$$(\text{Defect density in testing}) = \beta_0 + \beta_1 (\text{Years of programming experience})$$

the null hypothesis tested is $H_0: \beta_1 = 0$ with alternative hypothesis $H_a: \beta_1 \neq 0$.

Table 23 Regression Models for *Years of Experience*

Source		PSPa 9A	PSPa 10A	PSPb 9A	PSPb 10A
Model	DF	1	1	1	1
	SS	100.1	177.1	846.3	185.2
	MS	100.1	177.1	846.3	185.2
Error	DF	70	65	364	306
	SS	12289.7	5895.4	119685.5	42010.9
	MS	175.6	90.7	328.8	137.3
Total	DF	71	66	365	307
	SS	12389.9	6072.4	120531.8	42196.1
F Ratio		0.6	2.0	2.6	1.3
Prob > F		0.4527	0.1671	0.1095	0.2464
R^2_a		-0.0061	0.0142	0.0043	0.0011

The effect of *years of experience* on *defect density in testing* was shown not to be statistically significant for all of the data sets. This indicates that *years of experience* is not a useful predictor variable for *defect density in testing*.

The parameter estimates of the regression model for *years of experience*, and the associated standard errors, are listed in Table 24 for the data sets including outliers.

Table 24 Estimates for *Years of Experience*

Parameter	PSPa 9A (std err)	PSPa 10A (std err)	PSPb 9A (std err)	PSPb 10A (std err)
β_0 (Intercept)	10.96*** (2.98)	5.99** (2.25)	13.59**** (1.49)	9.79**** (1.03)
β_1	0.17 (0.22)	0.23 (0.17)	0.19 (0.12)	0.10 (0.09)

The regression models for the data sets excluding outliers are provided in Table 25. Outliers were identified with respect to *defect density in testing*; no outliers were identified for *years of experience*.

Table 25 Regression Models for *Years of Experience* Excluding Outliers

Source		PSPa 9A	PSPa 10A	PSPb 9A	PSPb 10A
Model	DF	1	1	1	1
	SS	282.5	14.9	5.1	1.2
	MS	282.5	14.9	5.1	1.2
Error	DF	68	62	346	295
	SS	6930.8	3411.3	36388.3	21607.6
	MS	101.9	55.0	105.2	73.2
Total	DF	69	63	347	296
	SS	7213.3	3426.1	36393.3	21608.8
F Ratio		2.8	0.3	0.05	0.02
Prob > F		0.1006	0.6052	0.8265	0.8994
R^2_a		0.0250	-0.0117	-0.0028	-0.0033

The regression results for the data sets excluding outliers are similar to those for the data sets with outliers. All of the analyses, including and excluding outliers, therefore indicate that *years of experience* is not a useful predictor variable for *defect density in testing*.

The parameter estimates of the regression model for *years of experience*, and the associated standard errors, are listed in Table 26 for the data sets excluding outliers.

Table 26 Estimates for *Years of Experience* Excluding Outliers

Parameter	PSPa 9A (std err)	PSPa 10A (std err)	PSPb 9A (std err)	PSPb 10A (std err)
β_0 (Intercept)	8.17*** (2.31)	6.50*** (1.84)	12.58**** (0.86)	9.29**** (0.77)
β_1	0.28 (0.17)	0.07 (0.14)	-0.02 (0.07)	-0.01 (0.07)

While some researchers have empirically found that *years of experience* are related to quality [Zhang 1999, 153; Takahasi and Kamayachi 1985], that was not the case in this analysis, although the results are similar to those of Wohlin and Wesslen for PSP [Wohlin and Wesslen 1998, 57; Wohlin 2004, 223]. When the studies finding experience a useful factor for less-experienced programmers were performed in the 1970s and 1980s, entry-level programmers were relatively unfamiliar with computers. The familiarity of the general population with computers has grown markedly over the last few decades. Students and entry-level programmers in the 1990s are likely to be well-acquainted with computers before beginning their professional careers. The consequence is that the operational definition of *years of experience* for programmers is likely to have shifted in the last three decades. The results of the earlier studies may have been valid and yet be irrelevant to today's population of "inexperienced" programmers.

This does not imply that experience does not affect ability. Holmes found that for his PSP data, collected over a seven year period, developer experience was a significant factor [Holmes 2003]. His results, however, apply to a single individual engaged in applying PSP as part of a systematic improvement program. They indicate that individual professionals can substantially improve their performance over time, but they cannot be generalized to *years of experience* across different individuals, which is the point of this analysis.

Industry studies of the effect of *years of experience* on quality typically use the average number of years for the team [Zhang 1999, 153]. In averaging experience across the team, related factors in assigning professionals to the team with diverse backgrounds may impact the operational meaning of "experience." The relevant factors may be related to a well-qualified team, with a variety of skills and capabilities, and *years of experience* may be confounded with other factors related to the skills of the team. *Years of experience* may therefore be a useful surrogate for team ability, even though it provides little insight with respect to the ability of the individual professionals comprising the team.

4.6.6 Number of Languages Known

While length of experience is usually considered a poor surrogate for ability, breadth of experience is considered a more realistic indicator [Curtis, Krasner, and Iscoe 1988; Curtis 1988, 289], particularly for programmers with three or fewer years of experience [Sheppard et al. 1979, 47]. The number of languages that a programmer has a working knowledge of may be considered a reasonable surrogate for breadth of experience.

The regression results for the effect of the *number of languages known* on *defect density in testing* are shown in Table 27. For the regression model:

(Defect density in testing) $= \beta_0 + \beta_1$ *(Number of languages known)*

the null hypothesis tested is H_0: $\beta_1=0$ with alternative hypothesis H_a: $\beta_1 \neq 0$.

Table 27 Regression Models for *Number of Languages Known*

Source		**PSPa 9A**	**PSPa 10A**	**PSPb 9A**	**PSPb 10A**
Model	DF	1	1	1	1
	SS	19.0	17.6	313.8	1127.2
	MS	19.0	17.6	313.8	1127.2
Error	DF	93	88	366	308
	SS	16804.3	9225.3	121034.5	40864.9
	MS	180.7	104.8	330.7	131.7
Total	DF	94	89	367	309
	SS	16823.2	9242.9	121348.3	41992.2
F Ratio		0.1	0.2	0.9	8.5
Prob > F		0.7468	0.6827	0.3306	0.0038
R^2_a		-0.0096	-0.0094	-0.0001	0.0237

The effect of the *number of languages known* on *defect density in testing* was shown to be statistically significant for only one of the four data sets: (PSPb, 10A, Outliers). This indicates that the *number of languages known* is unlikely to be a useful predictor variable for *defect density in testing*.

The parameter estimates of the regression model for the *number of languages known*, and the associated standard errors, are listed in Table 28 for the data sets including outliers.

Table 28 Estimates for *Number of Languages Known*

Parameter	PSPa 9A (std err)	PSPa 10A (std err)	PSPb 9A (std err)	PSPb 10A (std err)
β_0 (Intercept)	14.17**** (2.93)	8.16*** (2.24)	16.90**** (1.89)	7.48**** (1.31)
β_1	-0.17 (0.52)	0.16 (0.40)	-0.41 (0.43)	0.86** (0.29)

The regression models for the data sets excluding outliers are provided in Table 29. Outliers were identified with respect to *defect density in testing*; no outliers were identified with respect to the *number of languages known*.

Table 29 Regression Models for *Number of Languages Known* Excluding Outliers

Source		PSPa 9A	PSPa 10A	PSPb 9A	PSPb 10A
Model	DF	1	1	1	1
	SS	6.3	92.2	154.9	0.8
	MS	6.3	92.2	154.9	0.8
Error	DF	89	82	344	298
	SS	8670.0	3674.2	33154.8	22312.3
	MS	97.4	44.8	96.4	74.9
Total	DF	90	83	345	299
	SS	8676.3	3766.4	33309.7	22313.1
F Ratio		0.07	2.1	1.6	0.01
Prob > F		0.7993	0.1552	0.2057	0.9194
R^2_a		-0.0105	0.0126	0.0018	-0.0033

The ANOVA results for the data sets excluding outliers are similar to those for the data sets with outliers, differing only for the one case that was shown to be statistically significant

when outliers were included. The preponderance of the evidence therefore indicates that the *number of languages known* is unlikely to be a useful predictor variable for *defect density in testing*.

The parameter estimates of the regression model for *number of languages known*, and the associated standard errors, are listed in Table 30 for the data sets excluding outliers.

Table 30 Estimates for *Number of Languages Known* Excluding Outliers

Parameter	PSPa 9A (std err)	PSPa 10A (std err)	PSPb 9A (std err)	PSPb 10A (std err)
β_0 (Intercept)	11.93**** (2.22)	5.04** (1.50)	13.20**** (1.10)	9.32**** (0.03)
β_1	-0.10 (0.40)	0.38 (0.26)	-0.32 (0.24)	0.03 (0.25)

While breadth of experience may be superior to length of experience in determining ability, the *number of languages known* does not appear to be useful surrogate for ability.

4.6.7 Percent of Time Programming

The PSP student background questionnaire was revised in 1998. The question on *percent of time programming* in the previous year was revised to be more detailed, asking about software requirements, design, code, and test percentages (among other topics) separately. The instructions state that the total percentage need not sum to 100%. One consequence is that 116 of the students in the PSP repository have percentages of programming experience that sum to more than 100%. This was operationally resolved by using 100% for totals greater than 100%, but this measure may be systematically flawed for this analysis, and no firm conclusions should be reached.

The regression results for the effect of the *percent of time programming* in the previous year on *defect density in testing* are shown in Table 31. For the regression model:

(Defect density in testing) $= \beta_0 + \beta_1$ *(Percent of time programming)*

the null hypothesis tested is $H_0: \beta_1=0$ with alternative hypothesis $H_a: \beta_1 \neq 0$.

Table 31 Regression Models for *Percent of Time Programming*

Source		**PSPa 9A**	**PSPa 10A**	**PSPb 9A**	**PSPb 10A**
Model	DF	1	1	1	1
	SS	0.005	34.4	67.6	155.7
	MS	0.005	34.4	67.6	155.7
Error	DF	32	31	330	274
	SS	3653.4	2041.1	108002.2	35672.1
	MS	114.2	65.8	327.3	130.2
Total	DF	33	32	331	275
	SS	3653.4	1075.5	108069.8	35827.7
F Ratio		0.0000	0.5	0.2	1.2
Prob > F		0.9947	0.4749	0.6498	0.2751
R^2_a		-0.0313	-0.0151	-0.0024	0.0007

The effect of the *percent of time programming* in the previous year on *defect density in testing* was shown not to be statistically significant for all of the data sets. This indicates that the *percent of time programming* in the previous year is not a useful predictor variable for *defect density in testing*.

The parameter estimates of the regression model for the *percent of time programming* in the previous year, and the associated standard errors, are listed in Table 32 for the data sets including outliers.

Table 32 Estimates for *Percent of Time Programming*

Parameter	PSPa 9A (std err)	PSPa 10A (std err)	PSPb 9A (std err)	PSPb 10A (std err)
β_0 (Intercept)	11.64* (5.29)	9.86* (4.05)	16.25**** (2.48)	12.23**** (1.63)
β_1	0.0005 (0.08)	-0.04 (0.06)	-0.01 (0.03)	-0.02 (0.02)

The regression models for the data sets excluding outliers are provided in Table 33. Outliers were identified with respect to *defect density in testing*; no outliers were identified with respect to the *percent of time programming* in the previous year.

Table 33 Regression Models for *Percent of Time Programming* Excluding Outliers

Source		PSPa 9A	PSPa 10A	PSPb 9A	PSPb 10A
Model	DF	1	1	1	1
	SS	80.4	0.3	23.2	99.5
	MS	80.4	0.3	23.2	99.5
Error	DF	30	29	315	264
	SS	2003.6	910.8	32411.4	18567.2
	MS	66.8	31.4	102.9	70.3
Total	DF	31	30	316	265
	SS	2084.0	911.1	32434.6	18666.7
F Ratio		1.2	0.01	0.2	1.4
Prob > F		0.2812	0.9172	0.6356	0.2354
R^2_a		0.0065	-0.0341	-0.0025	0.0016

The regression results for the data sets excluding outliers are similar to those for the data sets with outliers. All of the analyses, including and excluding outliers, indicate that the *percent*

of time programming in the previous year is not a useful predictor variable for *defect density in testing*.

The parameter estimates of the regression models for the *percent of time programming* in the previous year, and the associated standard errors, are listed in Table 34 for the data sets excluding outliers.

Table 34 Estimates for *Percent of Time Programming* Excluding Outliers

Parameter	PSPa 9A (std err)	PSPa 10A (std err)	PSPb 9A (std err)	PSPb 10A (std err)
β_0 (Intercept)	14.22** (4.11)	5.36 (2.91)	11.75**** (1.44)	10.48**** (1.22)
β_1	-0.07 (0.06)	0.004 (0.04)	0.009 (0.02)	-0.02 (0.02)

Recent experience in programming is primarily a concern for learning curve effects associated with programming skills in general. Given the small size of the PSP assignments, it seems likely that the bulk of any learning curve effects associated with basic programming skills are concentrated in the first few assignments.

4.6.8 Programming Language

Although some researchers have noted that the *programming language* used does not appear to be significantly correlated with productivity or quality except at a gross level [DeMarco and Lister 1999, 32-33; Yang and Paradi 2004], others have found that programming language can have a significant impact on both. For example, in defining the backfiring technique for estimating function points based on lines of code, Jones found noticeable differences between different languages: it takes 128 lines of code in C to implement one

function point, but only 53 in C++ [Jones 1995]. Similarly, defect injection rates have been found to differ by two to three times between different programming languages [Phipps 1991, 351], although this may be driven by productivity differences between languages.

The most frequently used languages for PSPb are C, C++, Java, and Visual Basic. For PSPa, only two students used Visual Basic, and none used Java, so the only languages considered for the PSPa data set are C and C++. The *Each Pair* and *All Pairs* tests indicate that the means for the *programming language* are not significantly different, as illustrated by the comparison circles in Figure 8 for (PSPb, 9A, Outliers).

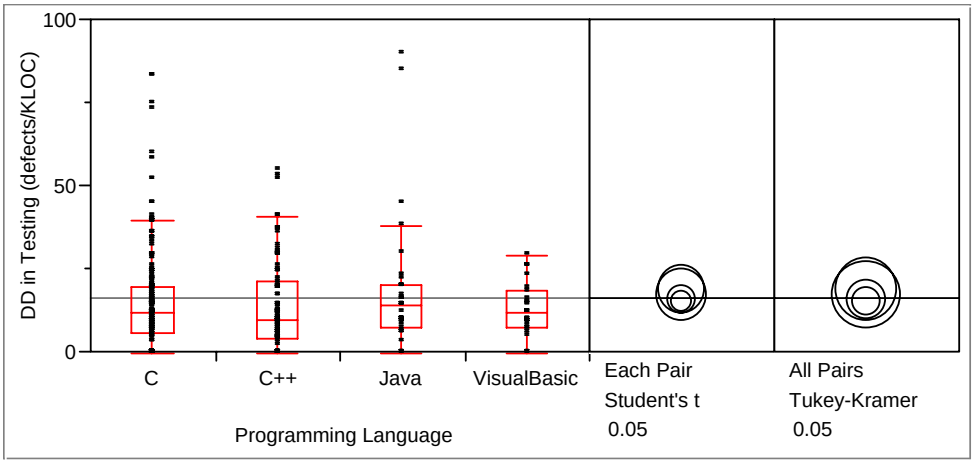

Figure 8 Differences Between *Programming Language* Levels

The ANOVA results for the effect of the *programming language* used on *defect density in testing* are shown in Table 35. The null hypothesis is $H_0 : \mu_C = \mu_{C++} = \mu_{Java} = \mu_{VisualBasic}$ for PSPb, and $H_0 : \mu_C = \mu_{C++}$ for PSPa, with alternative hypothesis H_a: *not all of the means are equal.*

Table 35 ANOVA for *Programming Language*

Source		PSPa 9A	PSPa 10A	PSPb 9A	PSPb 10A
Model	DF	1	1	3	3
	SS	26.5	45.8	342.2	14.8
	MS	26.5	45.8	114.1	4.9
Error	DF	72	66	285	238
	SS	14727.9	7447.0	109524.0	34141.9
	MS	204.6	112.8	384.3	143.5
Total	DF	73	67	288	241
	SS	14754.4	7492.9	109866.1	34156.7
F Ratio		0.1	0.4	0.3	0.03
Prob > F		0.7200	0.5261	0.8277	0.9915
R^2_a		-0.0121	-0.0089	-0.0074	-0.0122

The effect of the *programming language* used on *defect density in testing* was shown not to be statistically significant for all of the data sets. This indicates that the *programming language* used is not a useful predictor variable for *defect density in testing*.

The estimates of the means for *defect density in testing* at the different levels of the *programming language* used, and the associated standard errors for the means, are listed in Table 36 for the data sets including outliers. The model can be expressed as:

(*Defect density in testing*) = $\beta_{Programming\ Language}\ X_{Programming\ Language}$

where $\beta_{Programming\ Langue}$ is the level for the *programming language* used and $X_{Programming\ Language}$ is an indicator variable for whether that *programming language* is the correct one for the observation.

Table 36 Estimates for *Programming Language* Levels

Levels	PSPa 9A (std err)	PSPa 10A (std err)	PSPb 9A (std err)	PSPb 10A (std err)
C	13.80 (1.93)	9.68 (1.45)	15.69 (1.31)	10.85 (0.96)
C++	12.23 (2.67)	7.45 (2.53)	16.11 (2.49)	11.05 (1.63)
Java	---	---	19.13 (3.88)	11.63 (2.98)
Visual Basic	---	---	17.60 (5.12)	10.63 (2.19)

The ANOVA results for the data sets excluding outliers are provided in Table 37. Outliers were identified with respect to *defect density in testing*.

Table 37 ANOVA for *Programming Language* Excluding Outliers

Source		PSPa 9A	PSPa 10A	PSPb 9A	PSPb 10A
Model	DF	1	1	3	3
	SS	22.0	0.03	5.7	143.3
	MS	22.0	0.03	1.9	47.8
Error	DF	68	62	268	228
	SS	6555.8	3395.4	28534.4	15781.1
	MS	96.4	54.8	106.5	69.2
Total	DF	69	63	271	231
	SS	6577.8	3395.4	28540.1	15924.3
F Ratio		0.2	0.0006	0.02	0.7
Prob > F		0.6345	0.9811	0.9968	0.5590
R^2_a		-0.0113	-0.0161	-0.0110	-0.0040

The ANOVA results for the data sets excluding outliers are similar to those for the data sets with outliers. All of the analyses, including and excluding outliers, indicate that the *programming language* used is not a useful predictor variable for *defect density in testing*.

The estimates of the means for *defect density in testing* at the different levels of the *programming language* used, and the associated standard errors for the means, are listed in Table 38 for the data sets excluding outliers.

Table 38 Estimates for *Programming Language* Levels Excluding Outliers

Levels	PSPa 9A (std err)	PSPa 10A (std err)	PSPb 9A (std err)	PSPb 10A (std err)
C	10.79 (1.31)	7.40 (0.99)	12.66 (0.87)	9.66 (0.73)
C++	12.23 (2.67)	7.45 (2.53)	12.59 (1.26)	8.07 (0.99)
Java	---	---	13.11 (1.73)	9.22 (1.84)
Visual Basic	---	---	12.71 (1.59)	10.63 (2.19)

Language does not seem to affect quality when measured in terms of defect density. If defect density is fairly stable across languages, it seems reasonable to view lines of code as an appropriate measure of the opportunities for defects in a program.

Although language may not significantly affect defect density, the productivity difference between languages implies that the number of defects will vary significantly, depending on the language(s) used [Wesslen 1999, 225; Wohlin 2004, 223]. In other words, if it takes twice as many LOC to implement a program in language A as in language B, then there will be twice as many opportunities for defects in the language A program as for the language B program, even if the defect density is the same. Because of the impact of programming language on productivity, and the related effects on process variables such as review rates, the *programming language* used should not be ignored in analyzing software quality, although it may not be a statistically significant variable for quality (as measured by *defect density in testing*) when considered in isolation.

4.6.9 Discussion of the Potentially Confounding Variables

The results of the analyses of the potentially confounding variables indicate, with the exception of the PSP-specific factor of the *assignment*, that none of the variables is a potentially useful predictor variable. These results are summarized in Table 39.

Table 39 Statistically Significant Results for Potentially Confounding Variables

Variable	Including Outliers	Excluding Outliers
C1) *assignment* (9A vs 10A)	2 of 2	2 of 2
C2) *all ten* vs *less than ten* assignments finished	1 of 4	0 of 4
C3) *PSP class*	1 of 4	0 of 4
C4) *highest degree attained*	0 of 4	0 of 4
C5) *years of programming experience*	0 of 4	0 of 4
C6) *percent of time programming* in the previous year	0 of 4	0 of 4
C7) *number of languages known*	1 of 4	0 of 4
C8) *programming language* used	0 of 4	0 of 4

These results are consistent for data sets including and excluding outliers, but the potentially confounding variables will be explored further in more sophisticated statistical models in Chapter 7.

4.7 EXPLORING SOLUTION COMPLEXITY (PROGRAM SIZE)

4.7.1 Program Size and Defect Density in Testing

Hypothesis S1 is that *program size* is related to *defect density in testing*. Since all defect prediction models use *program size* as a predictor variable (and many use it as the only predictor variable), this variable is expected to be significant.

The regression results for the effect of *program size* on *defect density in testing* are shown in Table 40. For the regression model:

(Defect density in testing) $= \beta_0 + \beta_1$ *(Program size)*

the null hypothesis tested is $H_0: \beta_1 = 0$ with alternative hypothesis $H_a: \beta_1 \neq 0$.

Table 40 Regression Models for *Program Size*

Source		**PSPa 9A**	**PSPa 10A**	**PSPb 9A**	**PSPb 10A**
Model	DF	1	1	1	1
	SS	1367.0	602.3	16893.0	5988.0
	MS	1367.0	602.3	16893.0	5988.0
Error	DF	161	143	828	735
	SS	44124.1	17697.7	313333.8	143915.2
	MS	274.1	123.8	378.4	195.8
Total	DF	162	144	829	736
	SS	45491.1	18300.0	330226.8	149903.2
F Ratio		5.0	4.9	44.6	30.6
Prob > F		0.0269	0.0290	<0.0001	<0.0001
R^2_a		0.0240	0.0261	0.0500	0.0386

The effect of *program size* on *defect density in testing* was shown to be statistically significant for all of the data sets. This indicates that *program size* is a useful predictor variable for *defect density in testing* as expected.

The parameter estimates of the regression models for *program size*, and the associated standard errors, are listed in Table 41 for the data sets including outliers.

Table 41 Estimates for *Program Size*

Parameter	PSPa 9A (std err)	PSPa 10A (std err)	PSPb 9A (std err)	PSPb 10A (std err)
β_0 (Intercept)	18.22**** (2.42)	12.53**** (1.95)	24.92**** (1.37)	16.90**** (1.12)
β_1	-0.03* (0.01)	-0.02* (0.007)	-0.05**** (0.007)	-0.02**** (0.004)

The regression models for the data sets excluding outliers are provided in Table 42. Outliers were identified with respect to both *defect density in testing* and *program size*.

Table 42 Regression Models for *Program Size* Excluding Outliers

Source		PSPa 9A	PSPa 10A	PSPb 9A	PSPb 10A
Model	DF	1	1	1	1
	SS	334.5	21.1	2098.7	1576.0
	MS	334.5	21.1	2098.7	1576.0
Error	DF	149	136	748	659
	SS	16661.4	7198.0	86154.3	46445.1
	MS	111.9	52.9	115.2	70.5
Total	DF	150	137	749	660
	SS	16995.9	7219.1	88253.0	48021.1
F Ratio		3.0	0.4	18.2	22.4
Prob > F		0.0858	0.5293	<0.0001	<0.0001
R^2_a		0.0131	-0.0044	0.0225	0.0314

The regression results for the data sets excluding outliers differ from those for the data sets with outliers in two cases: (PSPa, 9A, NoOutliers) and (PSPa, 10A, NoOutliers) were no longer shown to be statistically significant. The preponderance of the evidence indicates that *program size* is a useful predictor variable for *defect density in testing*, but the PSPa results reinforce the observation that there is a great deal of individual variation between programmers and suggest that further study of the effect of *program size*, which is contained in Chapter 7, is appropriate.

The parameter estimates of the regression model for *program size*, and the associated standard errors, are listed in Table 43 for the data sets excluding outliers.

Table 43 Estimates for *Program Size* Excluding Outliers

Parameter	PSPa 9A (std err)	PSPa 10A (std err)	PSPb 9A (std err)	PSPb 10A (std err)
β_0 (Intercept)	14.73**** (1.94)	7.60**** (1.33)	17.30**** (0.95)	13.64**** (0.92)
β_1	-0.02 (0.01)	-0.003 (0.005)	-0.02**** (0.006)	-0.02**** (0.004)

Defect density in testing decreases as *program size* increases within the PSP context. With respect to hypothesis S1, it appears that *program size* is related to *defect density in testing*, but the relationship is weaker than might have been expected.

4.7.2 Program Size and the Number of Defects Removed in Testing

As illustrated in Figure 9 for (PSPb, 9A, NoOutliers), there are several $\frac{n}{x}$ patterns in the size versus defect density chart. This pattern arises because the number of *defects removed in testing*, which corresponds to "n," has a fairly small number of integer values – typically zero to six defects are found in testing. *Program size*, which corresponds to "x," can vary significantly.

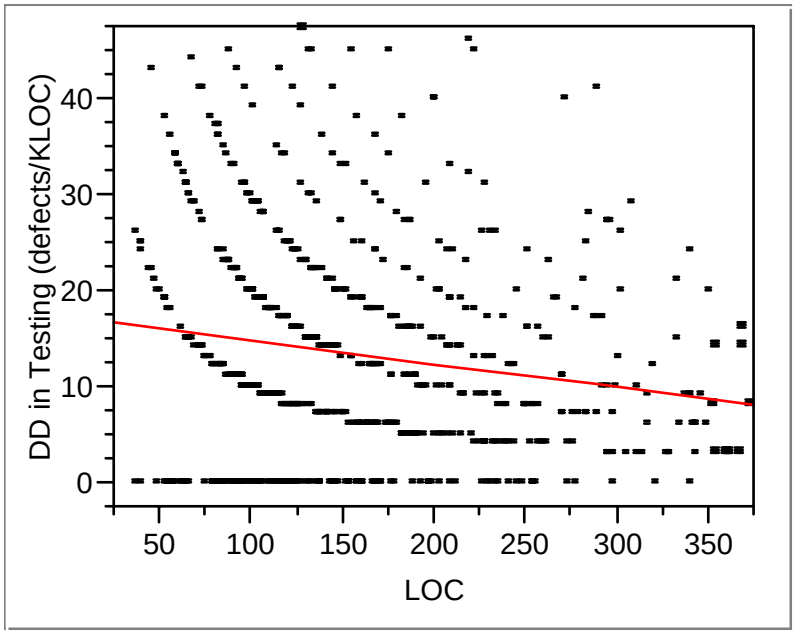

Figure 9 Regressing *Defect Density in Testing* on *Program Size*

Program size is already included in defect density as a normalizing factor. Regressing the number of *defects removed in testing* on the *program size* is a plausible alternative. The *programming language* used may affect the number of *defects removed in testing* since there are productivity differences between programming languages [Jones 1995], and it therefore becomes a useful criterion for splitting the data. This should not have been a concern in the analyses in Section 4.7.1 because of the use of defect density as the respondent variable. Because of the relatively small sizes of the resulting PSPa data sets, however, the analyses use only the PSPb data sets.

The regression results for the effect of *program size* on the number of *defects removed in testing* are shown in Table 44. For the regression model:

(Defects removed in testing) $= \beta_0 + \beta_1$ *(Program size)*

the null hypothesis tested is H_0: $\beta_1 = 0$ with alternative hypothesis H_a: $\beta_1 \neq 0$.

Table 44 Regressing *Defects Removed in Testing* on *Program Size*

Source		PSPb 9A C	PSPb 9A C++	PSPb 10A C	PSPb 10A C++
Model	DF	1	1	1	1
	SS	120.6	23.3	15.4	107.5
	MS	120.6	23.3	15.4	107.5
Error	DF	150	80	131	64
	SS	681.6	431.6	431.9	889.7
	MS	4.5	5.4	3.3	13.9
Total	DF	151	81	132	65
	SS	802.2	454.9	447.3	997.3
F Ratio		26.5	4.3	4.7	7.7
Prob > F		<0.0001	0.0409	0.0323	0.0071
R^2_a		0.1447	0.0393	0.0271	0.0939

The effect of *program size* on the number of *defects removed in testing* was shown to be statistically significant for all of the data sets. This indicates that *program size* is a useful predictor variable for the number of *defects removed in testing*.

The parameter estimates of the regression models for *program size*, and the associated standard errors, are listed in Table 45 for the data sets including outliers.

Table 45 Estimates for Regressing *Defects Removed in Testing* on *Program Size*

Parameter	PSPb 9A C (std err)	PSPb 9A C++ (std err)	PSPb 10A C (std err)	PSPb 10A C++ (std err)
β_0 (Intercept)	0.44 (0.36)	1.34** (0.49)	1.30**** (0.31)	0.26 (0.97)
β_1	0.01**** (0.002)	0.005* (0.002)	0.003* (0.001)	0.009** (0.003)

The regression models for the data sets excluding outliers are provided in Table 46. Outliers were identified with respect to both the number of *defects removed in testing* and *program size*.

Table 46 Regressing *Defects Removed in Testing* on *Program Size* Excluding Outliers

Source		PSPb 9A C	PSPb 9A C++	PSPb 10A C	PSPb 10A C++
Model	DF	1	1	1	1
	SS	8.8	0.3	0.7	0.1
	MS	8.8	0.3	0.7	0.1
Error	DF	136	73	122	58
	SS	221.4	150.3	251.2	117.1
	MS	1.6	2.1	2.1	2.0
Total	DF	137	74	123	59
	SS	230.2	150.7	252.0	117.3
F Ratio		5.4	0.2	0.4	0.05
Prob > F		0.0219	0.6935	0.5551	0.8219
R^2_a		0.0310	-0.0115	-0.0053	-0.0163

The regression results for the data sets excluding outliers differ from those for the data sets with outliers in three of the four cases: (PSPb, 9A, C++, NoOutliers), (PSPb, 10A, C, NoOutliers), and (PSP, 10A, C++, NoOutliers) were no longer shown to be statistically significant. These results suggest that, while *program size* should be considered a potential predictor variable for the number of *defects removed in testing* (or *defect density in testing*), the conclusion for individual performance is not the straightforward one that might be assumed based on prior research using project data.

The parameter estimates of the regression models for *program size*, and the associated standard errors, are listed in Table 47 for the data sets excluding outliers.

Table 47 Estimates for Regressing *Defects Removed in Testing* on *Program Size* Excluding Outliers

Parameter	PSPb 9A C (std err)	PSPb 9A C++ (std err)	PSPb 10A C (std err)	PSPb 10A C++ (std err)
β_0 (Intercept)	1.04*** (0.28)	1.53*** (0.39)	1.47*** (0.41)	1.85*** (0.49)
β_1	0.004* (0.002)	0.0009 (0.002)	0.001 (0.002)	-0.0004 (0.002)

The number of *defects removed in testing* increases as *program size* increases within the PSP context. This is as expected.

4.7.3 Discussion of Program Size

The preponderance of the evidence indicates that *program size* is related to software quality, although both *defect density in testing* and the number of *defects removed in testing* appear only weakly related to *program size* for the PSP data. For the reasons mentioned in

Section 4.3 and the loss of significance when outliers are excluded, the quality surrogate used in these analyses for overall quality is *defect density in testing*, although other quality surrogates are used in Section 4.8 and Chapter 8 as direct measures of the effectiveness of specific processes.

The poorness of the fit ($R^2_a<0.05$) is also somewhat surprising since many defect prediction models have been built based on *program size* with far better fits than this.

These observations reinforce the conclusion that the impact of individual differences on software quality overwhelms most other factors, although individual differences may be "smoothed out" by team performance in a project. The use of data from individual programmers, with significantly greater variation than that of project teams, is likely to have affected the goodness of the fit. Weinberg observed that the problem of ambiguous programming objectives allows the individual programmer to choose whether to emphasize program size, execution speed, clarity, development time, or other objectives, all of which leads to increased variability in program size [Weinberg 1998, 128-132].

4.8 EXPLORING THE PROCESS VARIABLES

The focus of my research is not specifically on the quality trend over PSP assignments. My research focuses on the fundamental process drivers for quality that the PSP assignments (and processes) are a surrogate for. Potential confounding variables were explored to ensure that they were appropriately addressed if necessary, and they will be revisited in Chapter 7. The specific hypotheses regarding primary variables for *disciplined processes* to be investigated involve the design, coding, and compilation processes. These hypotheses will be considered statistically significant for $\alpha=0.05$.

Hypothesis P1 is that *time per LOC in design* is related to *defect density in testing*. Insufficient time in design is a frequent complaint and can result in design work being performed less efficiently in later phases of the life cycle.

Hypothesis P2 is that *design review rate* is related to *defect density in testing*. A fast review rate leads to ineffective reviews.

Hypothesis P3 is that *defect density in design review* is related to *defect density in testing*.

Hypothesis P4 is that *time per LOC in coding* is related to *defect density in testing*.

Hypothesis P5 is that *code review rate* is related to *defect density in testing*.

Hypothesis P6 is that *defect density in code review* is related to *defect density in testing*.

Hypothesis P7 is that *defect density in compile* is related to *defect density in testing*.

Data sets are split by the *programming language* used. Since *programming language* affects process factors such as effort and is the primary technology factor available for consideration, using it as a splitting criterion is a conservative decision that should not adversely affect the statistical analyses given the wealth of data available, especially for PSPb. Since the process variables are being investigated, splitting by assignment would be inappropriate, given that the process systematically shifts by assignment.

Defect injection depends primarily on the competence of the programmer, on the quality of the inputs, and the production process. For example, the number of code defects depends on the coding skills of the programmer, the quality of the design, and the power of the design methods (as applied).

Reviews have an indirect impact, since the objective is to remove defects before they become visible in testing (and release). Defects can be removed from the software more efficiently early in the life cycle; at the end of development, the compiler acts as a verification

tool for the code identifying syntactical defects. A simple graphic of the cause-and-effect relationships is shown in Figure 10.

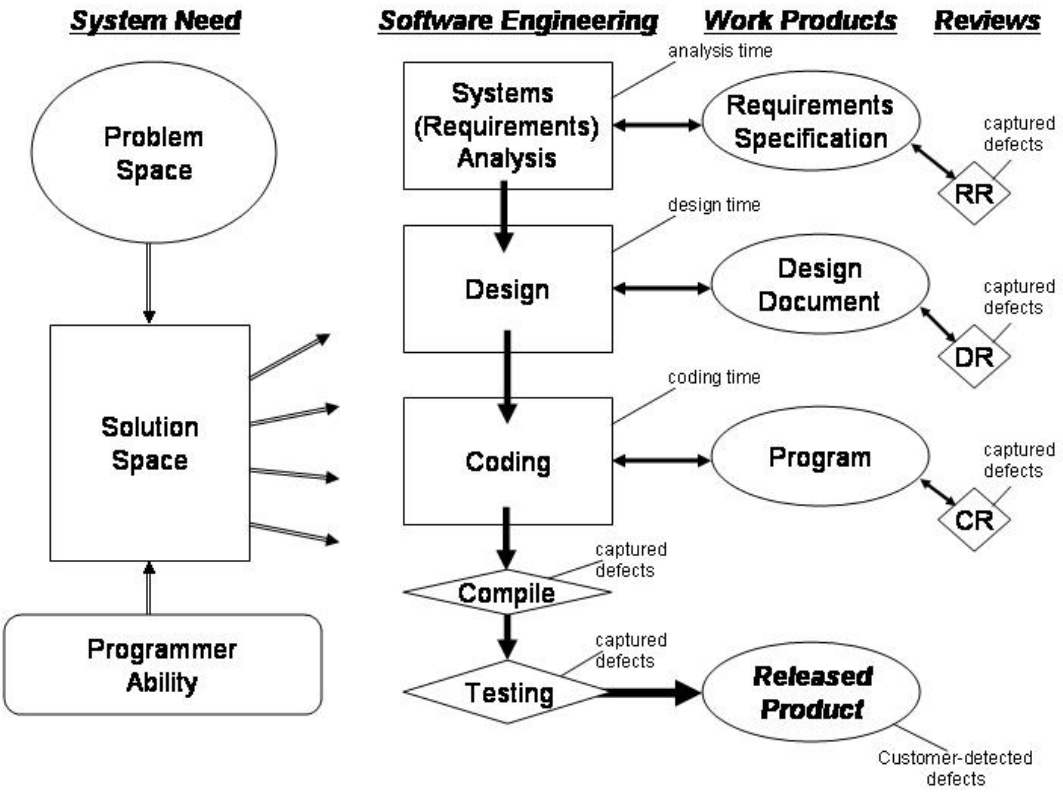

Figure 10 Dependencies Between Software Engineering Activities and Quality

As shown in the figure, the verification and validation activities in the diamond-shaped boxes are steps in the software process where defects may be identified and corrected. The number of *defects removed in testing* is a surrogate for quality, but even in a disciplined test process, defects may escape to the customer and cause failures in use. Testing can demonstrate the presence of defects; it cannot demonstrate their absence [Dijkstra 1979, 44]. The number of defects that can be found in testing will depend on the number of defects escaping from requirements analysis, design, coding, and compilation. Factors such as the confounding variables already considered may affect the number of defects injected in each phase of the life cycle. Process factors, such as review rates, will affect the defect removal effectiveness of

verification and validation activities such as reviews. Compiling the code can also be considered a verification step in the sense that syntactic defects will be identified by the compiler.

For the process variables analyzed in this section, outliers for the specific process variable are identified and excluded, along with the outliers for *defect density in testing*, in those analyses where outliers are excluded.

4.8.1 Design Time

The number of design defects that may impact *defect density in testing* will depend on the number that escape from design, which in turn depends on the quality of the requirements (the assignment statement), the number of design defects injected, and the defect removal effectiveness of the design reviews. There may also be interactions between factors: a large number of defects injected may be alleviated by skill in removing defects during design reviews or by the effectiveness of later steps in the process, such as the code reviews. Competing factors to consider in investigating design time include:

- Investing more time in the design process may correspond to increasing "reflective thought," which has been shown to be correlated with improved quality [Campbell 1999, 97]. In this context, more time in design presumably corresponds to more care in design decisions, leading to better quality by lowering the defect injection rate.
- A significant amount of reuse could result in less design time. It would also presumably be associated with a smaller program, and no defects would be added since otherwise the code would be modified rather than reused. This should not, however, affect defect density measures since the size of reused code is not recorded in the PSP repository.

- Longer design times may correspond to difficulties in design, which may in turn result from the student being an ineffective designer, perhaps due to unfamiliarity with the application domain or the design methods used in PSP. Short design times may indicate very efficient designers.
- Although the number of defects in design may be comparatively large if little time is spent in design, the student may remove defects in the coding (or code review) process effectively, if perhaps not as efficiently as could have been done in design.
- A small number of design defects could indicate an inadequate design with major problems to address. In this case, short design times may be associated with a large number of defects in coding, compile, or testing.

Design time per LOC is a leading indicator of quality, which could be used to control the design process since it can be measured in the design phase, if there is a dominant cause-and-effect relationship. An appropriate quality surrogate in this context is *design defect density*, i.e., the number of defects known to be present at the end of design (and beginning of design review, regardless of whether a design review was actually held) per KLOC. This enables a focus on the design process, regardless of defect removal activities that may occur later in the life cycle.

The regression results for the effect of *design time* on *design defect density* are shown in Table 48. For the regression model:

$$(\text{Design defect density}) = \beta_0 + \beta_1 (\text{Design time per LOC})$$

the null hypothesis tested is $H_0: \beta_1 = 0$ with alternative hypothesis $H_a: \beta_1 \neq 0$.

Table 48 Regressing *Design Defect Density* on *Design Time*

Source		PSPa C	PSPa C++	PSPb C	PSPb C++
Model	DF	1	1	1	1
	SS	48468.3	172847.2	61374.6	60835.5
	MS	48468.3	172847.2	61374.6	60835.5
Error	DF	675	138	1756	918
	SS	194000.1	151390.0	827849.6	584533.5
	MS	287.4	1097.0	471.4	636.7
Total	DF	676	139	1757	919
	SS	242468.3	324237.2	889224.2	645369.1
F Ratio		168.6	157.6	130.2	95.5
Prob > F		<0.0001	<0.0001	<0.0001	<0.0001
R^2_a		0.1987	0.5297	0.0685	0.0933

The effect of *design time* on *design defect density* was shown to be statistically significant for all of the data sets. This indicates that *design time* is a useful predictor variable for *design defect density*.

The parameter estimates of the regression model for *design time*, and the associated standard errors, are listed in Table 49 for the data sets including outliers.

Table 49 Estimates for Regressing *Design Defect Density* on *Design Time*

Parameter	PSPa C (std err)	PSPa C++ (std err)	PSPb C (std err)	PSPb C++ (std err)
β_0 (Intercept)	6.10**** (0.84)	1.38 (3.34)	7.57**** (0.69)	9.57**** (1.03)
β_1	11.36**** (0.87)	38.84**** (2.85)	10.72**** (0.94)	13.59**** (1.39)

The regression models for the data sets excluding outliers are provided in Table 50. Outliers were identified with respect to *design defect density* and *design time*.

Table 50 Regressing *Design Defect Density* on *Design Time* Excluding Outliers

Source		PSPa C	PSPa C++	PSPb C	PSPb C++
Model	DF	1	1	1	1
	SS	11670.7	1026.7	16737.8	14610.5
	MS	11670.7	1026.7	16737.8	14610.5
Error	DF	609	120	1559	809
	SS	82332.3	25426.2	191966.6	116236.2
	MS	135.2	211.9	123.1	143.7
Total	DF	610	121	1560	810
	SS	94003.0	26452.9	208704.4	130846.7
F Ratio		86.3	4.8	135.9	101.7
Prob > F		<0.0001	0.0296	<0.0001	<0.0001
R^2_a		0.1227	0.0308	0.0796	0.1106

The regression results for the data sets excluding outliers are similar to those for the data sets with outliers. All of the analyses, including and excluding outliers, indicate that *design time* is a useful predictor variable for *design defect density*.

The parameter estimates of the regression models for *design time*, and the associated standard errors, are listed in Table 51 for the data sets excluding outliers.

Table 51 Estimates for Regressing *Design Defect Density* on *Design Time* Excluding Outliers

Parameters	PSPa C (std err)	PSPa C++ (std err)	PSPb C (std err)	PSPb C++ (std err)
β_0 (Intercept)	4.47**** (0.75)	7.58*** (2.17)	3.18**** (0.47)	4.51**** (0.65)
β_1	12.60**** (1.36)	9.54* (4.33)	11.42**** (0.98)	15.92**** (1.58)

Design defect density increases as *design time* increases within the PSP context. There are two plausible explanations for this result. First, learning curve effects associated with the design techniques introduced in assignment 9 may be confounding the results. The PSP design techniques, although effective, are older than the object-oriented and pattern design methods used by today's programmers. The students may be struggling with applying these unfamiliar techniques and, as a consequence, be making more mistakes during design as they endeavor to use them. As a corollary, the problems in the PSP assignments are relatively simple, and the design process may have less effect on software quality than the coding process, which may compound the learning curve effects associated with unfamiliar design techniques. Second, it is possible that significant reuse, with defects already removed and whose size is not included in the *program size* measure, leads to smaller programs and less design time.

In the first case, the root cause is ineffective design; in the second, it is superior design. Interactions between *design time* and other variables are considered in Chapter 7 and may suggest which driver is predominant.

4.8.2 Design Review Rate

The effect of design and code reviews on software quality is indirect, although removing design and code defects early in the life cycle will decrease the number of defects available in testing. The effectiveness of the reviews cannot be directly measured until the end of the project, however, and the process control measures available are those associated with performing an effective review, e.g., the preparation and meeting review rates and the defects found by the reviews.

The review rate is the amount of time spent by a reviewer inspecting the design (or other work product) normalized by the size of the design (expressed in LOC for PSP). For PSP, since the review is for individuals, there is no meeting as would occur in an inspection, therefore the "review" rate is analogous to the preparation rate for inspections. A design review rate faster than 200 LOC/hour is considered unacceptable [Fagan 1976; Fagan 1986].

An appropriate quality surrogate in this context is *defect removal effectiveness*, i.e., the percentage of defects detected of those known to be present at the beginning of the design review. This is a direct measure of software quality resulting from the review.

If there are no known defects in the design at the time of the review (as determined at the end of the assignment), the effectiveness of the review is irrelevant. Assignments where no defects are available to be identified in a review are therefore excluded. Assignments where no design review was held are also excluded from this analysis.

For this analysis, the *design review rate* is measured in minutes/LOC rather than LOC/hour. Time normalized by size is similar to the other normalized measures and is used in later multiple regression models where there is a natural progression from "no reviews held" (0

minutes/LOC) to “fast review rates” (< 0.3 minutes/LOC, which corresponds to 200 LOC/hour) to “recommended review rates” (> 0.3 minutes/LOC).

The regression results for the effect of *design review rate* on *defect removal effectiveness* are shown in Table 52. For the regression model:

(Defect removal effectiveness) $= \beta_0 + \beta_1$ *(Design review rate)*

the null hypothesis tested is $H_0: \beta_1 = 0$ with alternative hypothesis $H_a: \beta_1 \neq 0$.

Table 52 Regressing *Defect Removal Effectiveness* on *Design Review Rate*

Source		**PSPa C**	**PSPa C++**	**PSPb C**	**PSPb C++**
Model	DF	1	1	1	1
	SS	4896.0	6.8	5259.2	6788.5
	MS	4896.0	6.8	5259.2	6788.5
Error	DF	145	32	372	247
	SS	225183.1	37897.3	533311.0	356060.7
	MS	1553.0	1184.3	1433.6	1401.1
Total	DF	146	33	373	248
	SS	230079.0	37904.1	538570.2	352849.2
F Ratio		3.2	0.006	3.7	4.8
Prob > F		0.0779	0.9402	0.0582	0.0286
R^2_a		0.0145	-0.0311	0.0071	0.0153

The effect of *design review rate* on *defect removal effectiveness* was shown to be statistically significant for one of the four data sets: (PSPb, C++, Outliers). This indicates that *design review rate* may be a useful predictor variable for *defect removal effectiveness*.

The parameter estimates of the regression models for *design review rate*, and the associated standard errors, are listed in Table 53 for the data sets including outliers.

Table 53 Estimates for Regressing *Defect Removal Effectiveness* on *Design Review Rate*

Parameter	PSPa C (std err)	PSPa C++ (std err)	PSPb C (std err)	PSPb C++ (std err)
β_0 (Intercept)	36.59**** (4.41)	50.49**** (6.80)	49.03**** (2.94)	46.73**** (3.38)
β_1	25.48 (14.35)	-0.67 (8.88)	16.52 (8.62)	22.36* (10.30)

The regression models for the data sets excluding outliers are provided in Table 54. Outliers were identified with respect to *design review rate*. Note that unusually large values of *design review rate* correspond to unusually meticulous design reviews in this particular instance (the interquartile limit for unusually "fast" *design review rates* is less than zero, therefore there can be no outliers for fast review rates).

Table 54 Regressing *Defect Removal Effectiveness* on *Design Review Rate* Excluding Outliers

Source		PSPa C	PSPa C++	PSPb C	PSPb C++
Model	DF	1	1	1	1
	SS	13602.4	10366.7	31626.8	7322.8
	MS	13602.4	10366.7	32626.8	7322.8
Error	DF	133	28	337	238
	SS	205697.5	25214.0	489333.0	334738.4
	MS	1546.6	900.5	1452.0	1406.5
Total	DF	134	29	338	239
	SS	219299.9	35580.7	520959.8	342061.2
F Ratio		8.8	11.5	21.8	5.2
Prob > F		0.0036	0.0021	<0.0001	0.0234
R^2_a		0.0550	0.2661	0.0579	0.0173

The regression results for the data sets excluding outliers differ from those for the data sets with outliers in three cases since all of the results for data sets excluding outliers were shown to be statistically significant. The preponderance of the evidence therefore indicates that *design review rate* is a useful predictor variable for *defect removal effectiveness*.

The parameter estimates of the regression model for *design review rate*, and the associated standard errors, are listed in Table 55 for the data sets excluding outliers.

Table 55 Estimates for Regressing *Defect Removal Effectiveness* on *Design Review Rate* Excluding Outliers

Parameter	PSPa C (std err)	PSPa C++ (std err)	PSPb C (std err)	PSPb C++ (std err)
β_0 (Intercept)	27.02**** (5.81)	18.56 (10.79)	36.50**** (4.11)	43.61**** (4.15)
β_1	91.84** (30.97)	154.57** (45.56)	82.00**** (17.57)	37.84* (16.58)

For all of the statistically significant data sets, including and excluding outliers, the greater the *design review rate,* as measured by minutes/LOC, the greater the *defect removal effectiveness* of the design reviews. Using the normal definition in software engineering of *design review rate* as LOC/hour, this is equivalent to saying the slower the design review, the greater the *defect removal effectiveness.*

It is worth investigating whether following recommended practice for the *design review rate* is effective. The recommended *design review rate* is less than 200 LOC/hour (or greater than 0.3 minutes/LOC). A faster rate is considered ineffective, and re-inspection should be scheduled. This provides two classes of design review based on review rate: those where the design review rate is faster than the recommended (specification) limit of 200 LOC/hour; and design reviews according to recommended practice. As illustrated in Figure 11 for (PSPb, C, Outliers), the classes of design review rate appear significantly different. Note that a "fast design review rate" corresponds to less than 0.3 minutes/LOC; the term "fast" is used here to connect with the normal software engineering usage.

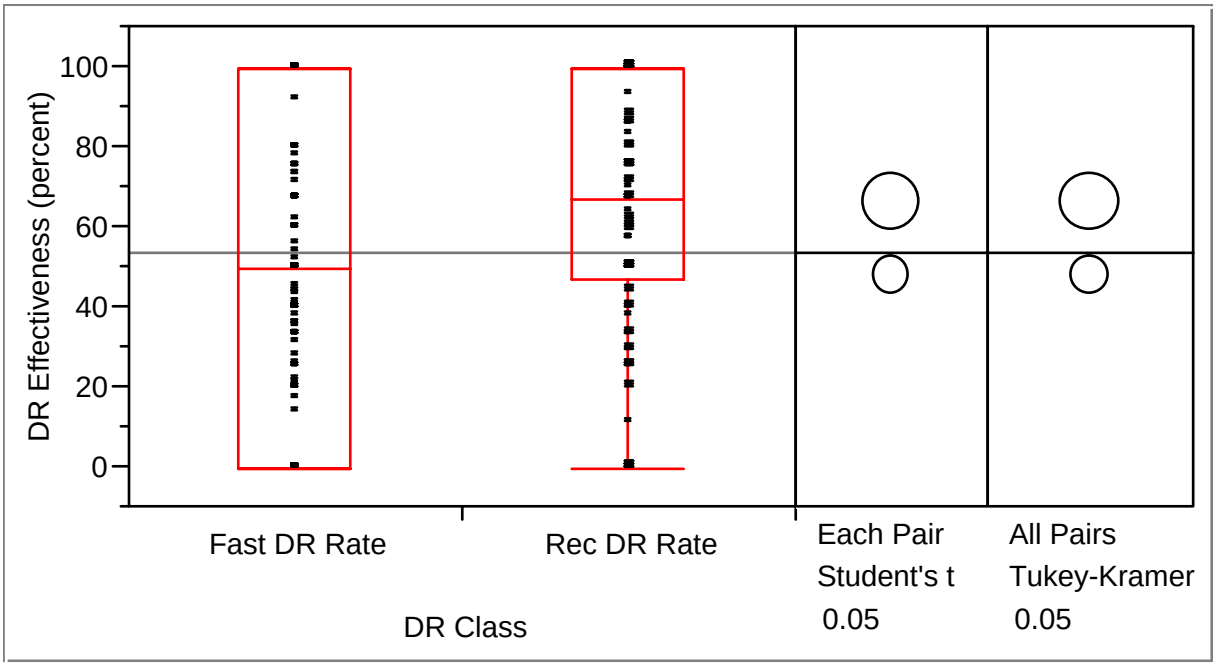

Figure 11 Differences in *Design Review Classes*

The ANOVA results for the effect of *design review class* on *defect removal effectiveness* are shown in Table 56. The null hypothesis is $H_0 : \mu_{FastDRRate} = \mu_{RecDRRate}$ with alternative hypothesis H_a: *not all of the means are equal*.

Table 56 ANOVA for Regressing *Defect Removal Effectiveness* on *Design Review Class*

Source		PSPa C	PSPa C++	PSPb C	PSPb C++
Model	DF	1	1	1	1
	SS	7326.6	2790.1	23420.8	2430.1
	MS	7326.6	2790.1	23420.8	2430.1
Error	DF	145	32	372	247
	SS	222752.4	35114.0	515149.4	350419.1
	MS	1536.2	1097.3	1384.8	1418.7
Total	DF	146	33	373	248
	SS	230079.0	37904.1	538570.2	352849.2
F Ratio		6.1[W]	2.5	19.2[W]	2.2[W]
Prob > F		0.0173[W]	0.1206	<0.0001[W]	0.1442[W]
R^2_a		0.0252	0.0447	0.0409	0.0029

The effect of *design review class* on *defect removal effectiveness* was shown to be statistically significant for two of the four data sets: (PSPa, C, Outliers) and (PSPb, C, Outliers). This indicates that *design review class* may be a useful predictor variable for *defect removal effectiveness*.

The estimates of the means for *defect removal effectiveness* at the different levels of *design review class*, and the associated standard errors for the means, are listed in Table 57 for the data sets including outliers. The model can be expressed as:

(*Defect removal effectiveness*) = $\beta_{DR\ Class}\, X_{DR\ Class}$

where $\beta_{DR\ Class}$ is the level for the *design review class* and $X_{DR\ Class}$ is an indicator variable for whether that *design review class* is the correct one for the observation.

Table 57 Estimates for Regressing *Defect Removal Effectiveness* on *Design Review Class*

Levels	PSPa C* (std err)	PSPa C++ (std err)	PSPb C**** (std err)	PSPb C++ (std err)
Fast DR Rate	38.69 (3.65)	44.80 (6.99)	48.15 (2.38)	49.77 (2.91)
Recommended DR Rate	57.48 (6.63)	65.33 (8.97)	66.57 (3.18)	56.92 (3.90)

The ANOVA results for the data sets excluding outliers are provided in Table 58. Outliers were identified with respect to *design review rate*.

Table 58 ANOVA for Regressing *Defect Removal Effectiveness* on *Design Review Class* Excluding Outliers

Source		PSPa C	PSPa C++	PSPb C	PSPb C++
Model	DF	1	1	1	1
	SS	6943.7	4213.5	27829.5	1080.6
	MS	6943.7	4213.5	27829.5	1080.6
Error	DF	133	28	337	238
	SS	212356.2	31367.2	493130.3	340980.6
	MS	1596.7	1120.3	1463.3	1432.7
Total	DF	134	29	338	239
	SS	219299.9	35580.7	520959.8	342061.2
F Ratio		4.3	3.8	19.0	1.0[W]
Prob > F		0.0389	0.0626	<0.0001	0.3287[W]
R^2_a		0.0244	0.0869	0.0506	-0.0010

The ANOVA results for the data sets excluding outliers are similar to those for the data sets with outliers; the same data sets were shown to have statistically significant results, and

(PSPa, C++, NoOutliers) is close to statistical significance. These results indicate that *design review class* may be a useful predictor variable for *defect removal effectiveness*.

The estimates of the means for *defect removal effectiveness* at the different levels of *design review class*, and the associated standard errors for the means, are listed in Table 59 for the data sets excluding outliers.

Table 59 Estimates for Regressing *Defect Removal Effectiveness* on *Design Review Class* Excluding Outliers

Levels	PSPa C* (std err)	PSPa C++ (std err)	PSPb C**** (std err)	PSPb C++ (std err)
Fast DR Rate	38.69 (3.65)	44.80 (6.99)	48.31 (2.38)	50.15 (2.91)
Recommended DR Rate	63.00 (10.13)	76.60 (10.12)	70.24 (4.24)	55.20 (4.25)

These results indicate that *design review rate* is related to *defect removal effectiveness*, but the results are mixed for whether a review rate less than 200 LOC/hour provides significantly better performance than a faster review rate.

4.8.3 Defect Density in Design Review

It is unclear how *defect density in testing* should relate to defect density as found in design reviews. A high defect density in the review could suggest a defect-prone module with many more defects to be found. It could also suggest a superior review, and that the majority of the defects were found, thus leading to a high-quality program.

If the review process is properly and consistently performed, then a reasonably predictable percentage of defects should be found by the review. To ensure that the reviews are properly performed, those that were outside the specification limits of 200 LOC/hour for design

reviews were excluded in the analyses below. Assignments where no design review was held were also excluded.

The regression results for the effect of *defect density in design review* on *defect density in testing* are shown in Table 60. For the regression model:

$$(\text{Defect density in testing}) = \beta_0 + \beta_1 (\text{Defect density in design review})$$

the null hypothesis tested is $H_0: \beta_1 = 0$ with alternative hypothesis $H_a: \beta_1 \neq 0$.

Table 60 Regression Models for *Defect Density in Design Review*

Source		**PSPa C**	**PSPa C++**	**PSPb C**	**PSPb C++**
Model	DF	1	1	1	1
	SS	609.1	972.4	3550.8	28293.9
	MS	609.1	972.4	3550.8	28293.9
Error	DF	180	36	618	333
	SS	66290.4	6330.6	215459.6	143957.2
	MS	368.3	175.9	348.6	432.3
Total	DF	181	37	619	334
	SS	66899.5	7303.1	219010.4	172251.1
F Ratio		1.7	5.5	10.2	65.4
Prob > F		0.2001	0.0243	0.0015	<0.0001
R^2_a		0.0036	0.1091	0.0146	0.1618

The effect of *defect density in design review* on *defect density in testing* was shown to be statistically significant for three of the four data sets. This indicates that *defect density in design review* should be a useful predictor variable for *defect density in testing*.

The parameter estimates of the regression models for *defect density in design review*, and the associated standard errors, are listed in Table 61 for the data sets including outliers.

Table 61 Estimates for *Defect Density in Design Review*

Parameter	PSPa C (std err)	PSPa C++ (std err)	PSPb C (std err)	PSPb C++ (std err)
β_0 (Intercept)	14.27**** (1.72)	8.27** (2.63)	13.85**** (0.88)	10.55**** (1.28)
β_1	0.15 (0.12)	0.14* (0.06)	0.14** (0.04)	0.38**** (0.05)

The regression models for the data sets excluding outliers are provided in Table 62. Outliers were identified with respect to *defect density in testing* and *defect density in design review*.

Table 62 Regression Models for *Defect Density in Design Review* Excluding Outliers

Source		PSPa C	PSPa C++	PSPb C	PSPb C++
Model	DF	1	1	1	1
	SS	63.6	85.0	401.6	599.2
	MS	63.6	85.0	401.6	599.2
Error	DF	173	34	563	298
	SS	28297.8	3872.9	89214.3	41250.6
	MS	163.6	113.9	158.5	138.4
Total	DF	174	35	564	299
	SS	28361.3	3957.9	89616.0	41849.8
F Ratio		0.4	0.7	2.5	4.3
Prob > F		0.5339	0.3938	0.1120	0.0383
R^2_a		-0.0035	-0.0073	0.0027	0.0110

The regression results for the data sets excluding outliers differ from those for the data sets with outliers in two cases: (PSPa, C++, NoOutliers) and (PSPb, C, NoOutliers) were no

longer shown to be statistically significant. The results indicate that *defect density in design review* may be a useful predictor variable for *defect density in testing*.

The parameter estimates of the regression models for *defect density in design review*, and the associated standard errors, are listed in Table 63 for the data sets excluding outliers.

Table 63 Estimates for *Defect Density in Design Review* Excluding Outliers

Parameter	PSPa C Estimate (std err)	PSPa C++ Estimate (std err)	PSPb C Estimate (std err)	PSPb C++ Estimate (std err)
β_0 (Intercept)	12.47**** (1.19)	8.48** (2.46)	12.37**** (0.65)	10.15**** (0.87)
β_1	0.06 (0.09)	0.08 (0.08)	0.08 (0.05)	0.14* (0.07)

Defect density in testing increases as *defect density in design review* increases within the PSP context. This may imply that low-quality designs correspond to low-quality software, although the interaction between the review rate and defect density should also be considered.

4.8.4 Coding Time

Coding time per LOC is a leading indicator of quality, which could be used to control the coding process since it can be measured in the code phase. The number of code defects that may impact *defect density in testing* will depend on the number that escape from coding, which in turn depends on the quality of the design, the number of code defects injected, and the defect removal effectiveness of the code reviews. There may be interactions between factors: a large number of defects injected may be alleviated by skill in removing defects during code reviews or by the effectiveness of later steps in the process, such as the compiler. Competing factors to consider in investigating *coding time* include:

- Longer *coding times* may correspond to doing design work in the coding step due to a poor design.
- A significant amount of reuse could result in less coding time. It would also presumably be associated with a smaller program, and no defects would be added since otherwise the code would be modified rather than reused. This should not, however, affect defect density measures since the size of reused code is not recorded in the PSP repository.
- Longer *coding times* may correspond to difficulties in coding, which may in turn result from the student being a poor coder. Short *coding times* may indicate very efficient coders.
- Short code (and code review) times may result in a small number of code defects, but they may indicate an inadequate program with major problems to address.

An appropriate quality surrogate in this context is *code defect density*, i.e., the number of defects known to be present at the end of coding (and beginning of code review, regardless of whether a code review was actually held) per KLOC.

The regression results for the effect of *coding time* on *code defect density* are shown in Table 64. For the regression model:

$$(\text{Code defect density}) = \beta_0 + \beta_1 (\text{Coding time per LOC})$$

the null hypothesis tested is $H_0: \beta_1 = 0$ with alternative hypothesis $H_a: \beta_1 \neq 0$.

Table 64 Regressing *Code Defect Density* on *Coding Time*

Source		PSPa C	PSPa C++	PSPb C	PSPb C++
Model	DF	1	1	1	1
	SS	2153740.4	21974.1	1436046.2	741084.8
	MS	2153740.4	21974.1	1436046.2	741084.8
Error	DF	675	138	1756	918
	SS	2093891.1	349998.7	7239537.5	360.1490.1
	MS	3102.0	2536.2	4123.0	3923.0
Total	DF	676	139	1757	919
	SS	4247631.4	371972.8	8675583.7	4342574.9
F Ratio		694.3	8.7	348.3	188.9
Prob > F		<0.0001	0.0038	<0.0001	<0.0001
R^2_a		0.5063	0.0523	0.1651	0.1698

The effect of *coding time* on *code defect density* was shown to be statistically significant for all of the data sets. This indicates that *coding time* is a useful predictor variable for *code defect density*.

The parameter estimates of the regression models for *coding time*, and the associated standard errors, are listed in Table 65 for the data sets including outliers.

Table 65 Estimates for Regressing *Code Defect Density* on *Coding Time*

Parameter	PSPa C (std err)	PSPa C++ (std err)	PSPb C (std err)	PSPb C++ (std err)
β_0 (Intercept)	23.71**** (2.81)	57.43**** (7.34)	49.71**** (2.23)	44.51**** (3.32)
β_1	52.03**** (1.97)	21.02** (7.14)	36.92**** (1.98)	49.24**** (3.58)

The regression models for the data sets excluding outliers are provided in Table 66. Outliers were identified with respect to *code defect density* and *coding time*.

Table 66 Regressing *Code Defect Density* on *Coding Time* Excluding Outliers

Source		PSPa C	PSPa C++	PSPb C	PSPb C++
Model	DF	1	1	1	1
	SS	73407.7	4329.9	345119.2	142331.2
	MS	73407.7	4329.9	345119.2	142331.2
Error	DF	603	123	1580	831
	SS	778914.1	160894.0	2351483.3	1455403.1
	MS	1291.7	1308.1	1488.0	1751.0
Total	DF	604	124	1581	832
	SS	852321.8	165223.9	2696602.5	1597734.2
F Ratio		56.8	3.3	231.9	81.3
Prob > F		<0.0001	0.0713	<0.0001	<0.0001
R^2_a		0.0846	0.0183	0.1274	0.0880

The regression results for the data sets excluding outliers differ from those for the data sets with outliers in only one case: (PSPa, C++, NoOutliers) was no longer shown to be

statistically significant. The preponderance of the evidence therefore indicates that *coding time* is a useful predictor variable for *code defect density*.

The parameter estimates of the regression models for *coding time,* and the associated standard errors, are listed in Table 67 for the data sets excluding outliers.

Table 67 Estimates for Regressing *Code Defect Density* on *Coding Time* Excluding Outliers

Parameters	PSPa C (std err)	PSPa C++ (std err)	PSPb C (std err)	PSPb C++ (std err)
β_0 (Intercept)	38.63**** (2.96)	55.40**** (6.68)	37.23**** (2.09)	42.83**** (3.08)
β_1	27.52**** (3.65)	15.12 (8.31)	42.21**** (2.77)	40.43**** (4.48)

Code defect density increases as *coding time* increases within the PSP context. A lack of effort in design can be expected to result in an increase in *coding time,* as design work is performed in a coding context, with implications for software quality. For the multiple regression models in Chapter 7, this shift may be observed in a statistically significant interaction between *design time* and *coding time*.

4.8.5 Code Review Rate

For this analysis, the *code review rate* is measured in minutes/LOC. Assignments where no defects are available to be identified in a code review were excluded. Assignments where no code review was held were also excluded from this analysis.

The regression results for the effect of *code review rate* on *defect removal effectiveness* are shown in Table 68. For the regression model:

(Defect removal effectiveness) $= \beta_0 + \beta_1$ *(Code review rate)*

the null hypothesis tested is $H_0: \beta_1=0$ with alternative hypothesis $H_a: \beta_1 \neq 0$.

Table 68 Regressing *Defect Removal Effectiveness* on *Code Review Rate*

Source		**PSPa C**	**PSPa C++**	**PSPb C**	**PSPb C++**
Model	DF	1	1	1	1
	SS	6272.2	875.0	28375.4	9914.4
	MS	6272.2	875.0	28375.4	9914.4
Error	DF	239	55	611	328
	SS	176949.4	38294.6	523083.5	254640.6
	MS	740.4	696.3	856.1	776.3
Total	DF	240	56	612	329
	SS	183221.5	39169.5	551459.0	264554.9
F Ratio		8.5	1.3	33.1	12.8
Prob > F		0.0039	0.2672	<0.0001	0.0004
R^2_a		0.0302	0.0046	0.0491	0.0345

The effect of *code review rate* on *defect removal effectiveness* was shown to be statistically significant for three of the four data sets: (PSPa, C, Outliers), (PSPb, C, Outliers), and (PSPb, C++, Outliers). This indicates that *code review rate* should be a useful predictor variable for *defect removal effectiveness*.

The parameter estimates of the regression models for *code review rate*, and the associated standard errors, are listed in Table 69 for the data sets including outliers.

Table 69 Estimates for Regressing *Defect Removal Effectiveness* on *Code Review Rate*

Parameter	PSPa C (std err)	PSPa C++ (std err)	PSPb C (std err)	PSPb C++ (std err)
β_0 (Intercept)	46.06**** (2.41)	51.29**** (4.65)	35.59**** (2.04)	39.97**** (2.28)
β_1	17.71** (6.08)	10.28 (9.17)	37.07**** (6.44)	23.06*** (6.45)

The regression models for the data sets excluding outliers are provided in Table 70. Outliers were identified with respect to *code review rate.*

Table 70 Regressing *Defect Removal Effectiveness* on *Code Review Rate* Excluding Outliers

Source		PSPa C	PSPa C++	PSPb C	PSPb C++
Model	DF	1	1	1	1
	SS	4991.7	3620.2	25900.3	8267.7
	MS	4991.7	3620.2	25900.3	8267.7
Error	DF	222	52	583	304
	SS	160486.7	35467.1	501211.4	230898.4
	MS	722.9	682.1	859.7	759.5
Total	DF	223	53	584	305
	SS	165478.5	39087.3	527111.8	239166.1
F Ratio		6.9	5.3	30.1	10.9
Prob > F		0.0092	0.0253	<0.0001	0.0011
R^2_a		0.0258	0.0752	0.0475	0.0314

The regression results for the data sets excluding outliers differ from those for the data sets with outliers in only one case: (PSPa, C++, NoOutliers) was shown to be statistically significant. All of the data sets were shown to be statistically significant for the data sets without

outliers. The preponderance of the evidence therefore indicates that *code review rate* is a useful predictor variable for *defect removal effectiveness*.

The parameter estimates of the regression models for *code review rate*, and the associated standard errors, are listed in Table 71 for the data sets excluding outliers.

Table 71 Estimates for Regressing *Defect Removal Effectiveness* on *Code Review Rate* Excluding Outliers

Parameter	PSPa C (std err)	PSPa C++ (std err)	PSPb C (std err)	PSPb C++ (std err)
β_0 (Intercept)	43.36**** (3.17)	39.94**** (1.23)	33.18**** (2.42)	36.18**** (3.09)
β_1	32.35** (12.31)	57.61* (25.01)	49.77**** (9.07)	41.30** (12.52)

Defect removal effectiveness increases as *code review rate* increases within the PSP context. Using the normal definition in software engineering of *code review rate* as LOC/hour, this is equivalent to saying the slower the code review, the greater the *defect removal effectiveness.*

It is worth investigating whether following recommended practice for the *code review rate* is effective. The recommended *code review rate* is less than 200 LOC/hour (or greater than 0.3 minutes/LOC). A faster rate is considered ineffective, and re-inspection should be scheduled. This provides two classes of code review based on review rate: those where the *code review rate* is faster than the recommended (specification) limit of 200 LOC/hour; and code reviews according to recommended practice.

The regression results for the effect of *code review class* on *defect removal effectiveness* are shown in Table 72. The null hypothesis against *defect removal effectiveness* is $H_0 : \mu_{FastCRRate} = \mu_{RecCRRate}$ with alternative hypothesis H_a: *not all of the means are equal*.

Table 72 ANOVA for Regressing *Defect Removal Effectiveness* on *Code Review Class*

Source		**PSPa C**	**PSPa C++**	**PSPb C**	**PSPb C++**
Model	DF	1	1	1	1
	SS	5692.5	3745.87	14509.4	5949.0
	MS	5692.5	3745.87	14509.4	5949.0
Error	DF	239	55	611	328
	SS	177529.1	35423.6	536949.6	258606.0
	MS	742.8	644.1	878.8	788.4
Total	DF	240	56	612	329
	SS	183221.5	39169.5	551459.0	264554.9
F Ratio		7.7	5.8	16.5	7.5
Prob > F		0.0061	0.0192	<0.0001	0.0063
R^2_a		0.0270	0.0792	0.0247	0.0195

The effect of *code review class* on *defect removal effectiveness* was shown to be statistically significant for all of the data sets including outliers. This indicates that *code review class* is a useful predictor variable for *defect removal effectiveness.*

The estimates of the means for *defect removal effectiveness* at the different levels for the two classes of *code review rate*, and the associated standard errors for the means, are listed in Table 73 for the data sets including outliers. The model can be expressed as:

(*Defect removal effectiveness*) = $\beta_{CR\ Class} X_{CR\ Class}$

where $\beta_{CR\ Class}$ is the level for the class of *code review rate* and $X_{CR\ Class}$ is an indicator variable for whether that *code review class* is the correct one for the observation.

Table 73 Estimates for Regressing *Defect Removal Effectiveness* on *Code Review Class*

Levels	PSPa C** (std err)	PSPa C++* (std err)	PSPb C**** (std err)	PSPb C++** (std err)
Fast CR Rate	47.93 (1.98)	48.53 (4.71)	41.90 (1.44)	43.51 (1.84)
Recommended CR Rate	58.88 (3.70)	65.33 (4.21)	52.42 (2.14)	53.22 (2.82)

The ANOVA results for the data sets excluding outliers are provided in Table 74. Outliers were identified with respect to *code review rate*.

Table 74 ANOVA for Regressing *Defect Removal Effectiveness* on *Code Review Class* Excluding Outliers

Source		PSPa C	PSPa C++	PSPb C	PSPb C++
Model	DF	1	1	1	1
	SS	4384.5	3780.8	10740.8	2442.5
	MS	4384.5	3780.8	10740.8	2442.5
Error	DF	222	52	583	304
	SS	161094.0	35306.6	516371.0	236723.5
	MS	725.7	679.0	885.7	778.7
Total	DF	223	53	584	305
	SS	165478.5	39087.3	527111.8	239166.1
F Ratio		6.0	5.6	12.1	3.1
Prob > F		0.0147	0.0221	0.0005	0.0775
R^2_a		0.0221	0.0794	0.0187	0.0070

The ANOVA results for the data sets excluding outliers differ from those for the data sets with outliers in only one case: (PSPb, C++, NoOutliers) was not shown to be statistically significant, although it is close. The preponderance of the evidence therefore indicates that *code review class* is a useful predictor variable for *defect removal effectiveness*.

The estimates of the means for *defect removal effectiveness* at the different levels for the two classes of *code review rate*, and the associated standard errors for the means, are listed in Table 75 for the data sets excluding outliers.

Table 75 Estimates for Regressing *Defect Removal Effectiveness* on *Code Review Class* Excluding Outliers

Levels	PSPa C* (std err)	PSPa C++* (std err)	PSPb C*** (std err)	PSPb C++ (std err)
Fast CR Rate	47.93 (1.98)	48.53 (4.71)	42.04 (1.45)	43.53 (1.84)
Recommended CR Rate	58.71 (4.23)	66.28 (4.90)	51.61 (2.32)	50.61 (3.09)

These analyses indicate that *code review rate* is related to *defect removal effectiveness*.

4.8.6 Defect Density in Code Review

It is unclear how *defect density in testing* should relate to defect density as found in code review. A high defect density in the review could suggest a defect-prone module with many more defects to be found. It could also suggest a superior review, and that the majority of the defects were found, thus leading to a high-quality program.

If the review process is properly and consistently performed, then a reasonably predictable percentage of defects should be found by the review. To ensure that the reviews were properly performed, those that were outside the specification limits of 200 LOC/hour for code reviews were excluded in the analyses below. Assignments where no code review was held were also excluded.

The regression results for the effect of *defect density in code review* on *defect density in testing* are shown in Table 76. For the regression model:

$$(\text{Defect density in testing}) = \beta_0 + \beta_1 (\text{Defect density in code review})$$

the null hypothesis tested is $H_0: \beta_1 = 0$ with alternative hypothesis $H_a: \beta_1 \neq 0$.

Table 76 Regression Models for *Defect Density in Code Review*

Source		PSPa C	PSPa C++	PSPb C	PSPb C++
Model	DF	1	1	1	1
	SS	8829.5	850.4	5503.8	46129.8
	MS	8829.5	850.4	5503.8	46129.8
Error	DF	249	55	634	336
	SS	82175.8	13184.4	217656.1	126177.7
	MS	330.0	239.7	343.3	375.5
Total	DF	250	56	635	337
	SS	91005.3	14034.8	223160.0	172307.5
F Ratio		26.8	3.5	16.0	122.8
Prob > F		<0.0001	0.0649	<0.0001	<0.0001
R^2_a		0.0934	0.0435	0.0231	0.2655

The effect of *defect density in code review* on *defect density in testing* was shown to be statistically significant for three of the four data sets: (PSPa, C, Outliers), (PSPb, C, Outliers), and (PSPb, C++, Outliers). (PSPa, C++, Outliers) is close. This indicates that *defect density in code review* should be a useful predictor variable for *defect density in testing*.

The parameter estimates of the regression models for *defect density in code review*, and the associated standard errors, are listed in Table 77 for the data sets including outliers.

Table 77 Estimates for *Defect Density in Code Review*

Parameter	PSPa C Estimate (std err)	PSPa C++ Estimate (std err)	PSPb C Estimate (std err)	PSPb C++ Estimate (std err)
β_0 (Intercept)	10.00**** (1.59)	9.87** (3.02)	12.53**** (1.03)	5.93**** (1.36)
β_1	0.20**** (0.04)	0.10 (0.05)	0.11**** (0.03)	0.32**** (0.03)

The regression models for the data sets excluding outliers are provided in Table 78. Outliers were identified with respect to *defect density in testing* and *defect density in code review*.

Table 78 Regression Models for *Defect Density in Code Review* Excluding Outliers

Source		PSPa C	PSPa C++	PSPb C	PSPb C++
Model	DF	1	1	1	1
	SS	1213.6	2.7	854.8	1778.2
	MS	1213.6	2.7	854.8	1778.2
Error	DF	226	52	588	305
	SS	31419.1	6746.1	95785.5	39436.5
	MS	139.0	129.7	162.9	129.3
Total	DF	227	53	589	306
	SS	32632.8	6748.8	96640.2	41214.7
F Ratio		8.7	0.02	5.2	13.8
Prob > F		0.0035	0.8859	0.0223	0.0002
R^2_a		0.0329	-0.0188	0.0072	0.0400

The regression results for the data sets excluding outliers are similar to those for the data sets including outliers. The preponderance of the evidence therefore indicates that *defect density in code review* is a useful predictor variable for *defect density in testing*.

The parameter estimates of the regression models for *defect density in code review*, and the associated standard errors, are listed in Table 79 for the data sets excluding outliers.

Table 79 Estimates for *Defect Density in Code Review* Excluding Outliers

Parameter	PSPa C Estimate (std err)	PSPa C++ Estimate (std err)	PSPb C Estimate (std err)	PSPb C++ Estimate (std err)
β_0 (Intercept)	9.56**** (1.26)	12.45**** (2.59)	11.78**** (0.81)	8.28**** (1.03)
β_1	0.13** (0.04)	-0.009 (0.06)	0.06* (0.03)	0.14*** (0.04)

Defect density in testing increases as *defect density in code review* increases within the PSP context. Code reviews may be identifying a high percentage of syntactical defects, such as the compiler might find, rather than more serious logical defects, which surface in testing. Wesslen found that PSP reviews identified a higher percentage of compile defects than design defects, suggesting that further improvement in the review process is needed for the code reviews to be as effective as they could be [Wesslen 1999, 32].

4.8.7 Defect Density in Compile

The compiler can act as a debugging tool in detecting syntactic defects, therefore the number of defects removed during compilation seems likely to be correlated to the number that will be removed during testing. The defects found by the compiler are relatively minor compared to those found during testing since more than five times the effort is required to repair

testing defects than compilation defects, but studies have shown that clusters of minor defects are highly correlated with major defects [Endres and Rombach 2003, 131-133; Glass 2004, 135-137].

The regression results for the effect of *defect density in compile* on *defect density in testing* are shown in Table 80. For the regression model:

(Defect density in testing) $= \beta_0 + \beta_1$ *(Defect density in compile)*

the null hypothesis tested is $H_0: \beta_1 = 0$ with alternative hypothesis $H_a: \beta_1 \neq 0$.

Table 80 Regression Models for *Defect Density in Compile*

Source		**PSPa C**	**PSPa C++**	**PSPb C**	**PSPb C++**
Model	DF	1	1	1	1
	SS	483963.5	35048.0	499387.9	234907.9
	MS	483963.5	35048.0	499387.9	234907.9
Error	DF	675	138	1756	918
	SS	1096141.7	76167.0	2211308.0	961010.7
	MS	1624.0	551.9	1259.0	1047.2
Total	DF	676	139	1757	919
	SS	1580105.2	111215.0	2710695.9	1195918.6
F Ratio		298.0	63.5	3.4	224.4
Prob > F		<0.0001	<0.0001	<0.0001	<0.0001
R^2_a		0.3053	0.3102	0.1838	0.1955

The effect of *defect density in compile* on *defect density in testing* was shown to be statistically significant for all of the data sets. This indicates that *defect density in compile* is a useful predictor variable for *defect density in testing*.

The parameter estimates of the regression model for *defect density in compile*, and the associated standard errors, are listed in Table 81 for the data sets including outliers.

Table 81 Estimates for *Defect Density in Compile*

Parameter	PSPa C (std err)	PSPa C++ (std err)	PSPb C (std err)	PSPb C++ (std err)
β_0 (Intercept)	13.66**** (1.83)	8.92** (3.02)	18.61**** (1.07)	13.48**** (0.31)
β_1	0.44**** (0.03)	0.44**** (0.06)	0.30**** (0.01)	0.31**** (0.02)

The ANOVA statistics for the data sets excluding outliers are provided in Table 82. Outliers were defined with respect to *defect density in compile* and *defect density in testing*.

Table 82 ANOVA for *Defect Density in Compile* Excluding Outliers

Source		PSPa C	PSPa C++	PSPb C	PSPb C++
Model	DF	1	1	1	1
	SS	24635.6	6462.7	72009.1	30370.8
	MS	24635.6	6462.7	72009.1	30370.8
Error	DF	608	123	1576	855
	SS	205182.5	28147.4	567025.4	278028.9
	MS	337.5	228.8	359.8	325.2
Total	DF	609	124	1577	856
	SS	229818.1	34610.1	639034.4	308399.6
F Ratio		73.0	28.2	200.1	93.4
Prob > F		<0.0001	<0.0001	<0.0001	<0.0001
R^2_a		0.1057	0.1801	0.1121	0.0974

The regression results for the data sets excluding outliers are similar to those for the data sets with outliers. All of the analyses, including and excluding outliers, indicate that *defect density in compile* is a useful predictor variable for *defect density in testing*.

The parameter estimates of the regression model for *defect density in compile*, and the associated standard errors, are listed in Table 83 for the data sets excluding outliers.

Table 83 Estimates for *Defect Density in Compile* Excluding Outliers

Parameter	PSPa C (std err)	PSPa C++ (std err)	PSPb C (std err)	PSPb C++ (std err)
β_0 (Intercept)	14.81**** (1.11)	10.76**** (2.18)	15.89**** (0.71)	15.15**** (0.85)
β_1	0.25**** (0.03)	0.26**** (0.05)	0.23**** (0.02)	0.13**** (0.01)

Defect density in testing increases as *defect density in compile* increases within the PSP context.

Although *defect density in compile* is a leading indicator for *defect density in testing*, it is of limited value in controlling the development process since it is measured so near testing. The earlier in the life cycle that a factor can be measured and used for decision making, the more useful it is. For example, design measures, if sufficiently correlated to *defect density in testing*, will be of more value than code measures, which in turn are more useful than compilation measures.

Adding later measures to a multiple regression model that spans the life cycle should, however, add value to models that are refined as a project continues. A model that includes compile-time data may be used to set expectations for the number of *defects removed in testing*. Defect estimation models are based on defects found in testing (which is later in the life cycle

than compile) and are used in planning testing and making product release decisions [AIAA R-013]. Defect prediction models are process-based, and useful for controlling the development process as well as making release decisions. Both kinds of models can be useful for process control, but the added insight possible with process-based models must be balanced with the additional difficulty and expense necessary to collect detailed process data.

4.8.8 Discussion of the Process Variables

The results for the process variables are summarized in Table 84, identifying which process variables should be considered in further analyses of the explanatory variables for software quality. As expected, the process variables are consistently related to software quality – more so than *program size*, which is the most common predictor variable in defect prediction models. Following "best practices," such as recommended review rates, is supported.

Table 84 Statistically Significant Results for the Process Variables

Process Variable (Quality Surrogate)	Including Outliers	Excluding Outliers
P1) *design time* (*design defect density*)	4 of 4	4 of 4
P2) *design review rate* (*defect removal effectiveness*)	1 of 4	4 of 4
P3) *defect density in design review* (*defect density in testing*)	3 of 4	1 of 4
P4) *coding time* (*code defect density*)	4 of 4	3 of 4
P5) *code review rate* (*defect removal effectiveness*)	3 of 4	4 of 4
P6) *defect density in code review* (*defect density in testing*)	3 of 4	3 of 4
P7) *defect density in compile* (*defect density in testing*)	4 of 4	4 of 4

These results indicate that the process variables are potentially useful predictor variables for software quality – and that a disciplined software process that consistently follows recommended practice is important for building high-quality software products. The process variables, and their interactions, will be explored further in more sophisticated statistical models in Chapter 7.

The R^2_a values are too low for the simple regression models to be of practical value in predicting defects. This does not imply that programmers could not build simple regression models useful for predicting their individual performance. This observation applies to simple regression models based on individual data for many different individuals, where the individual differences dominate other factors.

My analyses differ from those of previous studies of software projects in two crucial and related ways. First, the analyses are based on individual performance rather than project or team performance. Second, the amount of the variability in individual performance that is explained by simple regression models is far less than that reported for projects and teams. These results dramatically reinforce the observation that team performance is typically both better and less variable than individual performance since the R^2_a values for these models are notably less than those reported for team/project performance in the published literature.

4.9 EXPLORING PROGRAMMER ABILITY

Programmer ability can be inferred if there is a consistent pattern in the PSP assignments indicating that the programmers with superior performance on one assignment tend to be superior performers on other assignments. Possible explanations that rely on characterizations such as “the best designers may,” such as the discussion in Section 4.8.1, can be addressed, at

least in part, by a variable that empirically captures programmer ability. Hypothesis A1 is that relative *programmer ability* is consistent across PSP assignments as measured by average *defect density in testing* for assignments 1-3.

4.9.1 Comparing Improvement of Top and Bottom Quartiles

A statistically rigorous analysis of the impact of programmer ability would be based on natural growth curves or repeated measures ANOVA. That analysis is deferred to Chapter 7; for this simple analysis, *defect density in testing* was averaged for the first three assignments and used to identify the top, middle two, and bottom quartiles for student performance. Figure 12 illustrates the differences in performance for these three categories for the (PSPb, C) data set across the ten PSP assignments.

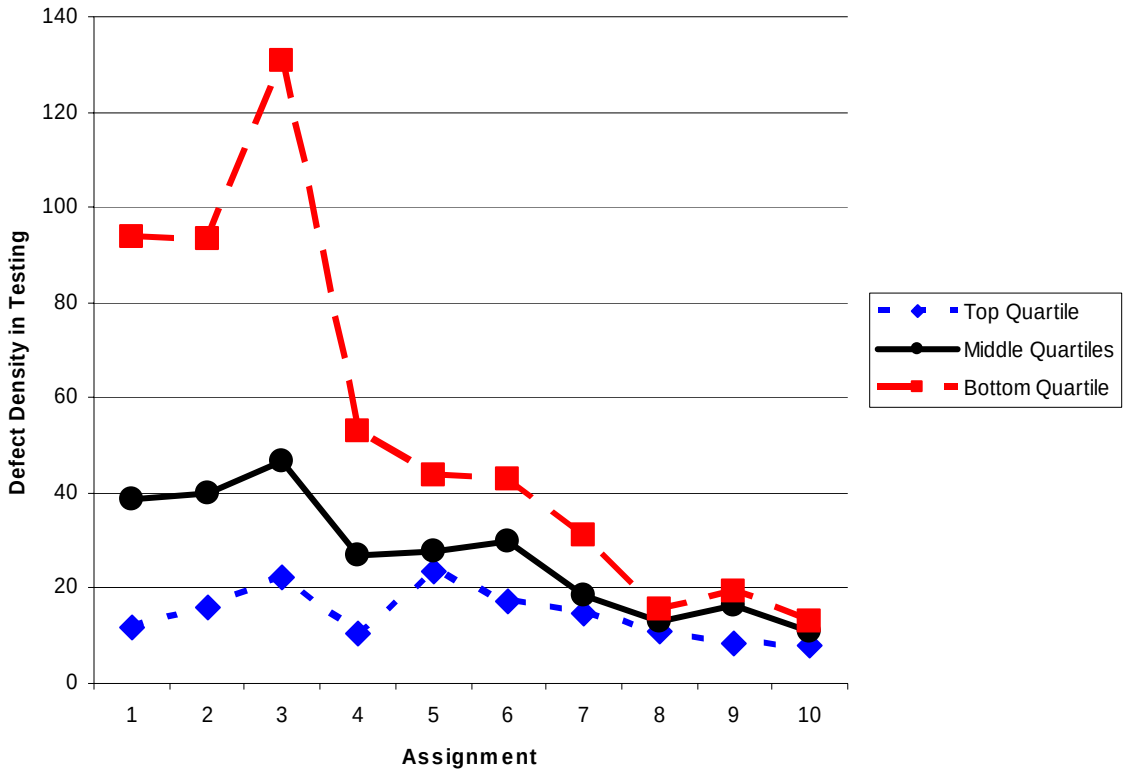

Figure 12 Trends for *Programmer Quartiles*

The students who are the top-quartile performers on the first three assignments tend to remain top performers (with the smallest *defect density in testing*) in the later assignments; the middle performers tend to remain in the middle; and the bottom-quartile performers (with the largest *defect density in testing*) tend to remain at the bottom. (A spike in the data for assignment 3 is consistently observed for all measures, suggesting that assignment 3 is somewhat more complex than the norm for the PSP assignments.)

The average performance and the standard deviation for the students in the top and bottom quartiles is contrasted for assignments 1 and 10 in Table 85 using the PSPa and PSPb data sets. Only students finishing all ten assignments are used in these calculations to ensure comparability between the two assignments.

Table 85 Comparing Top and Bottom Quartile Average Performance

	PSPa TQ : BQ	PSPb TQ : BQ	PSPa TQ : BQ	PSPb TQ : BQ
	Average		Standard Deviation	
1A	9.0 : 59.3	13.8 : 78.5	9.5 : 37.8	11.9 : 51.1
10A	3.5 : 17.1	6.6 : 15.6	3.5 : 17.9	8.5 : 12.9
N	20 : 26	144 : 134	20 : 26	144 : 134
Percent Change	61% : 71%	52% : 80%	63% : 53%	29% : 75%

The average performance of the top-quartile students improved 52-61%, and the bottom-quartile students improved 71-80%. The variability in the performance of the top-quartile students decreased 29-63%, and the variability of the bottom-quartile students decreased 53-75%.

To express this another way, top-quartile students improved by a factor between 2.1 and 2.6, and their variability decreased by a factor between 1.4 and 2.7. Bottom-quartile students improved by a factor between 3.5 and 5.0, and their variability decreased by a factor between 2.1 and 4.0. The performance of top-quartile students is better than that of bottom-quartile students by a factor of 5.7 to 6.6 initially, and changes to 2.4 to 4.9 by the end of PSP.

The comparisons between top and bottom performers for the data sets excluding outliers are provided in Table 86. Outliers were defined with respect to *defect density in testing* within the top and bottom quartiles for assignments 1 and 10. Note that *N* now differs for the top and bottom quartiles, even though only students finishing all ten assignments were considered.

Table 86 Comparing Top and Bottom Performers Excluding Outliers

	PSPa TQ : BQ	PSPb TQ : BQ	PSPa TQ : BQ	PSPb TQ : BQ
	Average		Standard Deviation	
1A	9.0 : 59.3	13.0 : 73.7	9.5 : 37.8	10.7 : 43.5
10A	3.5 : 13.3	6.1 : 15.2	3.5 : 12.3	6.8 : 11.9
N for 1A	20 : 26	141 : 130	20 : 26	141 : 130
N for 10A	20 : 24	143 : 133	20 : 24	143 : 133
Percent Change	61% : 78%	53% : 79%	63% : 67%	36% : 73%

When outliers are excluded, the top-quartile students improved 53-61%; bottom-quartile students improved 78-79%. The variability in the performance of the top-quartile students decreased 36-63%; the variability of the bottom-quartile students decreased 57-73%.

Top-quartile students improved by a factor between 2.1 and 2.6, and their variability decreased by a factor between 1.6 and 2.7. Bottom-quartile students improved by a factor between 4.6 and 4.8, and their variability decreased by a factor between 3.1 and 3.7. These values are similar to those when outliers were included: by adopting disciplined processes, top-quartile students improve their software quality by a factor more than two, and bottom-quartile students improve theirs by a factor more than four.

When outliers are excluded, the performance of top-quartile students is better than that of bottom-quartile students by a factor of 5.7 to 6.6 initially, and changes to 2.5 to 3.8 by the end of PSP. While top-quartile students continue to perform better than those in the bottom quartile, a disciplined process leads to superior performance for the bottom-quartile students, and even the top-quartile students improved markedly.

4.9.2 Comparing Top and Bottom Performers at the End of PSP

The differences between the top and bottom performers appear to persist across the course of the PSP class. This can be confirmed by comparing performance in the later assignments in the class. This also removes the need to restrict the data sets to those students who finished all ten assignments. As illustrated in Figure 13 for (PSPb, C, 9A), differences in performance for the first three assignments tend to persist in the later assignments.

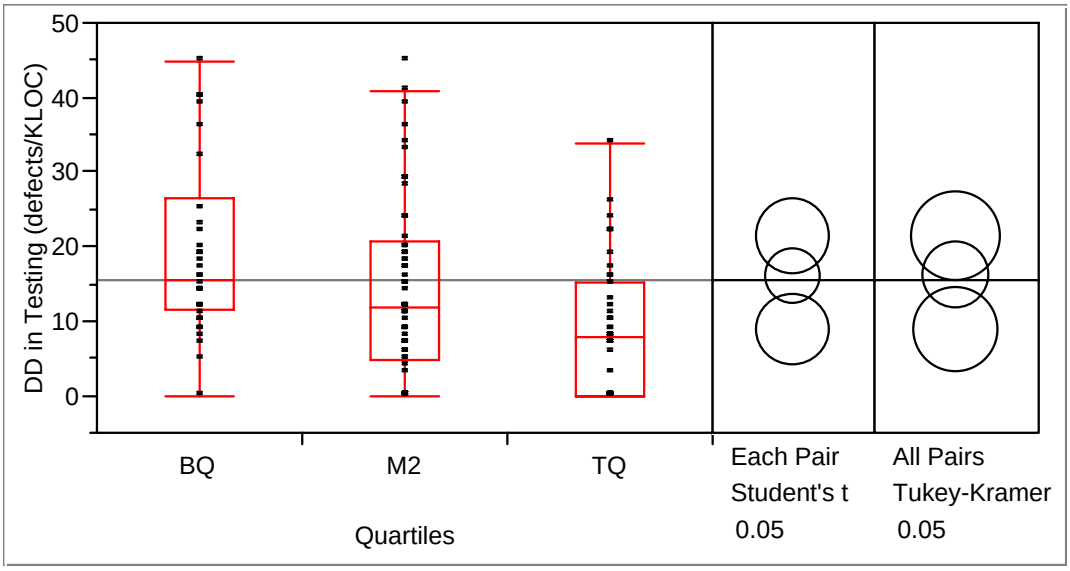

Figure 13 Differences in Performance Across Quartiles for (PSPb, C, 9A)

The *Each Pair* and *All Pairs* tests indicate that the means for the top and bottom quartiles of *programmer ability* are significantly different from each other, although there is overlap with the middle two quartiles.

The ANOVA results for the effect of *programmer quartile* on *defect density in testing* are shown in Table 87. The null hypothesis against *defect density in testing* is

$H_0 : \mu_{TQ} = \mu_{M2} = \mu_{BQ}$ with alternative hypothesis H_a: *not all of the means are equal.*

Table 87 ANOVA for *Programmer Quartile*

Source		PSPa 9A C	PSPa 9A C++	PSPb 10A C	PSPb 10A C++
Model	DF	2	2	2	2
	SS	3196.8	2021.1	349.3	810.6
	MS	1598.4	1010.6	174.7	405.3
Error	DF	149	79	130	63
	SS	36219.7	39228.9	15853.7	10572.2
	MS	243.1	496.6	122.0	167.8
Total	DF	151	81	132	65
	SS	39416.5	41250.0	16203.0	11382.9
F Ratio		10.1[W]	2.0	1.4	4.2[W]
Prob > F		0.0001[W]	0.1375	0.2425	0.0302[W]
R^2_a		0.0688	0.0249	0.0065	0.0417

The effect of *programmer quartile* on *defect density in testing* was shown to be statistically significant for only two of the four data sets. This indicates that *programmer quartile* may be a useful predictor variable for *defect density in testing.*

The estimates of the means for *defect density in testing* at the different levels of *programmer quartile*, and the associated standard errors for the means, are listed in Table 88 for the data sets including outliers. The model can be expressed as:

$$(\textit{Defect density in testing}) = \beta_{\textit{Programmer Quartile}} X_{\textit{Programmer Quartile}}$$

where $\beta_{\textit{Programmer Quartile}}$ is the level for *programmer quartile* and $X_{\textit{Programmer Quartile}}$ is an indicator variable for whether that *programmer quartile* is the correct one for the observation.

Table 88 Estimates for *Programmer Quartile* Levels

Levels	PSPb 9a C**** (std err)	PSPb 9A C++ (std err)	PSPb 10A C (std err)	PSPb 10A C++* (std err)
TQ	21.68 (2.76)	23.09 (5.62)	13.39 (1.62)	10.33 (4.14)
M2	16.35 (2.09)	18.10 (3.85)	10.72 (1.53)	13.82 (2.50)
BQ	9.14 (1.36)	8.61 (2.13)	8.73 (1.51)	5.84 (1.24)

The ANOVA results for the data sets excluding outliers are provided in Table 89. Outliers were identified with respect to *defect density in testing* for the top and bottom quartiles for assignments 1 and 10.

Table 89 ANOVA for *Programmer Quartile* Excluding Outliers

Source		PSPa 9A C	PSPa 9A C++	PSPb 10A C	PSPb 10A C++
Model	DF	2	2	2	2
	SS	1213.5	573.9	330.6	101.6
	MS	606.8	287.0	165.3	50.8
Error	DF	140	75	127	57
	SS	14240.4	9014.9	8524.4	2730.6
	MS	101.7	120.2	67.7	47.9
Total	DF	142	77	128	59
	SS	15453.9	9588.9	8855.0	2832.2
F Ratio		6.0	2.4	2.4	1.1
Prob > F		0.0033	0.0988	0.0910	0.3530
R^2_a		0.0654	0.0348	0.0221	0.0020

The ANOVA results for the data sets excluding differ from those for the data sets with outliers in one case: (PSPb, 10A, C++) is no longer shown to be significant. While the evidence continues to indicate that *programmer quartile* may be a useful predictor variable for *defect density in testing*, it does not appear to be a good distinguisher when disciplined processes are used.

The estimates of the means for *defect density in testing* at the different levels of *programmer quartile*, and the associated standard errors for the means, are listed in Table 90 for the data sets excluding outliers.

Table 90 Estimates for *Programmer Quartile* Excluding Outliers

Levels	PSPb 9a C** (std err)	PSPb 9A C++ (std err)	PSPb 10A C (std err)	PSPb 10A C++ (std err)
TQ	17.18 (1.71)	16.56 (4.35)	12.57 (1.44)	7.25 (3.13)
M2	12.58 (1.32)	13.80 (1.61)	8.80 (1.00)	8.73 (1.25)
BQ	9.14 (1.36)	8.61 (2.13)	8.73 (1.51)	5.84 (1.24)

It is intuitively clear that *programmer ability* is a fundamental driver for software quality, but a simple measure such as the quartiles for the initial PSP assignments may not be an adequate choice to characterize that ability.

4.9.3 Using a Continuous Measure of Programmer Ability

A better surrogate for *programmer ability* than the *programmer quartile* may be the direct use of the *average defect density in testing* for the first three assignments. Note that large values for this surrogate correspond to relatively low ability (PSP students may be above average for software professionals in general).

The regression results for the effect of *programmer ability*, as measured by the average *defect density in testing* on the first three assignments, on *defect density in testing* in assignments 9 and 10 are shown in Table 91. For the regression model:

$$\textit{(Defect density in testing)} = \beta_0 + \beta_1 \textit{(Programmer ability)}$$

the null hypothesis tested is $H_0: \beta_1 = 0$ with alternative hypothesis $H_a: \beta_1 \neq 0$.

Table 91 Regression Models for *Programmer Ability*

Source		PSPa 9A C	PSPa 9A C++	PSPb 10A C	PSPb 10A C++
Model	DF	1	1	1	1
	SS	3584.0	3369.5	573.1	160.7
	MS	3584.0	3369.5	573.1	160.7
Error	DF	150	80	131	64
	SS	35832.5	37880.5	15629.9	11222.1
	MS	238.9	473.5	119.3	175.3
Total	DF	151	81	132	65
	SS	39416.5	41250.0	16203.0	11382.9
F Ratio		15.0	7.1	4.8	0.9
Prob > F		0.0002	0.0092	0.0302	0.3419
R^2_a		0.0849	0.0702	0.0280	-0.0013

The effect of *programmer ability* on *defect density in testing* was shown to be statistically significant for three of the data sets. This indicates that *programmer ability* should be a useful predictor variable for *defect density in testing*.

The parameter estimates of the regression model for *programmer ability*, and the associated standard errors, are listed in Table 92 for the data sets including outliers.

Table 92 Estimates for *Programmer Ability*

Parameter	PSPa 9A C (std err)	PSPa 9A C++ (std err)	PSPb 10A C (std err)	PSPb 10A C++ (std err)
β_0 (Intercept)	9.65**** (2.00)	7.63 (3.99)	8.22**** (1.53)	8.82** (2.84)
β_1	0.12*** (0.03)	0.19** (0.07)	0.05* (0.02)	0.05 (0.05)

The ANOVA statistics for the data sets excluding outliers are provided in Table 93. Outliers were defined with respect to *defect density in testing*.

Table 93 ANOVA for *Programmer Ability* Excluding Outliers

Source		PSPa 9A C	PSPa 9A C++	PSPb 10A C	PSPb 10A C++
Model	DF	1	1	1	1
	SS	1207.3	896.7	534.1	0.5
	MS	1207.3	896.7	534.1	0.5
Error	DF	141	76	127	58
	SS	14246.6	8692.2	8320.9	2831.6
	MS	101.0	114.4	65.5	48.8
Total	DF	142	77	128	59
	SS	15453.9	9588.9	8855.0	2832.2
F Ratio		11.9	7.8	8.2	0.01
Prob > F		0.0007	0.0065	0.0050	0.9165
R^2_a		0.0716	0.0816	0.0529	-0.0171

The regression results for the data sets excluding outliers are similar to those for the data sets with outliers. The preponderance of the evidence therefore indicates that *programmer*

ability should be a useful predictor variable for *defect density in testing*, and that a continuous variable such as the average *defect density in testing* is preferable over a categorical variable, at least from a statistical perspective.

The parameter estimates of the regression model for *programmer ability*, and the associated standard errors, are listed in Table 94 for the data sets excluding outliers.

Table 94 Estimates for *Programmer Ability* Excluding Outliers

Parameter	PSPa 9A C (std err)	PSPa 9A C++ (std err)	PSPb 10A C (std err)	PSPb 10A C++ (std err)
β_0 (Intercept)	9.09^{****} (1.33)	7.53^{***} (2.18)	7.11^{****} (1.14)	7.48^{****} (1.57)
β_1	0.07^{***} (0.02)	0.12^{**} (0.04)	0.05^{**} (0.02)	0.003 (0.03)

Defect density in testing increases as *programmer ability*, as measured by average *defect density in testing* on the first three assignments, increases within the PSP context. To say this in a more intuitive manner, as ability improves, so does software quality.

4.9.4 Discussion of Programmer Ability

My research finds that *programmer ability*, as empirically measured by the *average defect density in testing* for the first three PSP assignments, should be a predictor variable for software quality. This holds true even for a simple regression model that does not address *program size* or any of the process variables that were also shown to be useful predictor variables.

A continuous measure is preferable to a categorical measure within the PSP context. There is a caveat in generalizing this observation, however. The use of measures such as defect

density for promotions and raises will drive dysfunctional behavior unless used in the context of a balanced set of project and organizational measures [Austin 1996]. Collecting this data for strictly informational purposes in building defect prediction models should not cause dysfunctional behavior, but there is a risk that collecting this kind of data will be perceived as being used for motivational purposes, which could in turn lead to dysfunctional behavior that might comprise the validity of the data. Categories such as *programmer quartile* may be less amenable to abuse and other factors could be incorporated that may be important in a project or organizational context – some of which may not be easily quantifiable, such as good teamwork skills.

While top-quartile students performed better than those in the bottom quartile on average, a disciplined process leads to significantly better performance for the bottom-quartile students, and even the top-quartile students improved markedly. Over the course of PSP, top-quartile students improved their software quality by a factor more than two, and bottom-quartile students improved theirs by a factor more than four. Variation in performance within each quartile also decreased markedly.

4.10 CONCLUSIONS FOR EXPLANATORY VARIABLES FOR SOFTWARE QUALITY

In this exploratory data analysis I found that 1) process-based variables are important factors for software quality; 2) *program size*, which is an indicator of solution complexity, is an important quality factor; and 3) *programmer ability* is an important quality factor when empirically measured. Other variables that may appear to be plausible surrogates for important areas such as ability and technology were not shown to be significant.

Building on earlier research into defect prediction models [Boehm et al. 2000, 254-268; Devnani-Chulani 1999; Evanco and Lacovava 1994; Neufelder 2000; Schneberger 1997; Wohlin and Wesslen 1998; Zhang 1999], the relevant explanatory variables for individual performance that are good candidates for use in defect prediction are *program size*, *programmer ability*, *design time*, *design review rate*, *defect density in design review, coding time*, *code review rate*, *defect density in code review*, and *defect density in compile*. The *PSP major process* and PSP *assignment* may capture additional variation due to application domain or learning curve effects, but they are of no practical value outside the PSP context. While the variables found to be important factors for software quality in my research are likely to also be important for software projects in general, other variables outside the scope of this analysis, such as those related to problem complexity or team effects, are likely to be important for projects in industry.

My research differs from previous PSP analyses in several respects. Hayes and Over focused on the impact of PSP at the *PSP major process* or *assignment* level [Hayes and Over 1997]. Their results have been replicated [Wohlin and Wesslen 1998; Wesslen 2000; Wohlin 2004], and PSP is widely considered an effective method for building high-quality software. My research replicates their results with respect to software quality, then explores the impact on software quality of more detailed process measures, such as design time, review rate, and defect density in reviews.

I was able to extend the research into the impact of potential confounding variables, such as academic degrees and experience, and rule out those factors. I was able to empirically consider the issue of *programmer ability* and verify its importance and influence on the various process variables. The issue of relative ability of programmers is particularly important, since the finding that even top-quartile performers improve over 2X refutes those who resist the need

for discipline, while acknowledging that their performance is superior and their opportunities for improvement are less than many of their colleagues.

My research supports the premise of PSP and similar process improvement strategies: disciplined software processes, such as PSP2 and PSP3, result in superior performance compared to *ad hoc* processes, such as PSP0. This improvement can be seen in both improved performance and decreased variation. It can be inferred that this is the minimum level of improvement that can be expected for a set of programmers since other researchers have observed that improvement continues after the PSP class [Hayes 1998; Ferguson et al. 1997; Holmes 2003; Hirmanpour and Schofield 2003].

My contribution in this analysis therefore consists of the following results:

- Disciplined processes were shown to improve individual performance in software quality by a factor of about five, similar to the results of previous researchers analyzing PSP data [Hayes and Over 1997, 22; Wesslen 2000; Wohlin 2004].
- Individual differences of more than an order-of-magnitude were shown to remain even when disciplined processes were used [Ferguson et al. 1997; Hayes and Over 1997, 22; Hayes 1998; Hirmanpour and Schofield 2003; Holmes 2003; Wohlin 2004, 212].
- *Programmer ability* was shown to significantly affect software quality when empirically measured; surrogates such as *years of experience* were not found to be useful, although some earlier researchers have found team-based experience significant [Takahashi and Kamayachi 1985; Zhang 1999].
- Top-quartile performers were shown to improve by a factor of two or more; bottom-quartile performers were shown to improve by a factor of four or more.

- Although technology factors, i.e., *programming language used*, may affect productivity, they were not shown to affect software quality as measured by *defect density in testing*, unlike some earlier researchers in a project/team environment [Gaffney 1984; Lipow 1982].
- *Program size* was shown to be a weak predictor of quality in the presence of individual differences, unlike the findings of most previous researchers in a project/team environment [Akiyama 1972; Compton and Withrow 1990; Criscione, Ferree, and Porter 2001; Halstead 1977, 87-91; Jones 1996; Lipow 1982; Lyu 1996; Fenton and Neil 1999; Fenton and Ohlsson 2000; Putnam and Myers 1997, 32].
- Detailed process measures provide more insight into performance than broad categories such as *PSP major process* or CMM maturity level, partially addressing Fenton and Neil's desire for more complete models [Fenton and Neil 1999, 153], at least within the context of individual programmers.

The practical implications of my research for software managers and professionals are relatively simple, although they may be challenging to address. First, although *programmer ability* is a crucial factor affecting software quality, surrogates such as seniority and academic credentials are inadequate for ranking programmers, and empirical measures that are more effective may cause dysfunctional behavior when used for determining raises and promotions [Austin 1996]. Second, consistent performance of recommended engineering practices improves software quality, even for top performers, who may resist discipline and measurement-based decisions because they already are superior performers. Third, organizations, teams, and individuals can use frameworks such as the Software CMM, TSP, and PSP to help

institutionalize disciplined, measurement-driven processes that are more effective than intuitively managed processes.

5.0 IDENTIFYING OUTLIERS IN THE SOFTWARE PROCESS

5.1 THE RESEARCH QUESTION: IDENTIFYING OUTLIERS

The research in this chapter focuses on identifying atypical programs. The common cause system for PSP software development can be characterized with respect to a number of factors, especially process variables, as has been shown in Chapter 4. While the PSP process is rigorously defined, it may not be consistently implemented (or implemented according to recommended practice). The primary emphasis in this chapter is on identifying atypical performance (outliers) and characterizing the stable process, i.e., the consistent and predictable process.

From an analytic perspective, atypical data should be discarded. To understand the impact of *process discipline* on individual performance, the common cause system must be characterized so that analyses focus on expected performance and is not skewed by a handful of atypical results. Identifying atypical performance in the software process or atypical entities in software work products is important for statistically analyzing process and product data and for statistical process control of the software process. Outliers may skew the results of a statistical analysis, but outliers that are not clearly erroneous should neither be completely discarded nor blindly included in a statistical analysis [Judd and McClelland 1989, 210; Neter et al. 1996, 103-104].

Outliers may result from three general sources [Judd and McClelland 1989, 207-237]. First are errors in data entry, which clearly should be corrected or discarded (if they can be

accurately identified). Invalid PSP data was identified and removed in Section 3.5 Second are outliers when the data sets are heterogeneous – they contain two or more separate types of entity. For process control, the traditional technique for identifying outliers is the control chart (or process behavior chart), which can be used to identify signals of assignable (or special) causes of variation in the common cause system used to build software. In the case of the control chart, the two different types of entity are the assignable causes and the common cause system. Causal analysis of the signals allows software professionals to take corrective and preventive actions as appropriate. Third are distributions with thick tails, in which "atypical" events may be relatively common.

In previous studies of individual performance, a few individuals were unable to solve the problems posed. These cases were considered atypical, although as many as one in six programmers might be involved, and their data was discarded in identifying order of magnitude differences in performance [Curtis 1988, 290]. This is similar to removing data for students not finishing all ten PSP assignments, but the removal of outliers allows an arguably more refined insight into normal performance.

The normal tool used for identifying special (or assignable) causes of variation in a process is the control chart. There are other techniques for identifying outliers; a corollary to this analysis is comparing a simple outlier identification technique (interquartile limits) with the XmR control chart.

5.2 IDENTIFYING THE COMMON CAUSE SYSTEM

The common cause system to be analyzed in depth is that for the final PSP processes, as captured in C and C++ programs for assignments 9 and 10 in the PSPb data set. The processes used are the epitome of the disciplined PSP, including reviews and design templates.

Process control techniques, such as control charts, enable in-process control, once the factors affecting software quality are known. In a retrospective analysis such as this one, the objective is to identify outliers that are atypical of normal performance and that can be excluded from further analysis. The *X* and *Xbar* charts are robust in the presence of non-normal data, therefore they are appropriate control charts to consider for identifying outliers given the skewed nature of software data [Schilling and Nelson 1976; Wheeler 2000].

For some measures, specification limits for factors such as review rates, in conjunction with the control limits, determine whether a process can be considered "capable." If the control limits that identify outliers are outside the specifications, then the process is not capable in the SPC sense, and the points outside the specification limits should be viewed as nonconformant to good practice.

Control charts may not be useful in the context of software processes. If the control limits are too wide to be useful, they do not add value. "Useful" suggests that some points are identified as signals, therefore causal analysis and corrective action can be taken. "Useful" may be a subjective judgment to some degree, but if the control limits for preparation rate or meeting rate are outside the specification limits for the rate, the direct conclusion is that the control chart does not add value.

A stable process, i.e., one from which assignable causes of variation have been removed, is predictable, and process performance is consistent over time. Predictable performance,

however, could be predictably bad. This is one of the concerns expressed by skeptics regarding process discipline and SPC. If the process is not capable, then the value added by SPC for control is negligible, but the need for improvement has been identified and quantified – a notable benefit in itself.

5.3 SPECIFICATION LIMITS FOR SOFTWARE PROCESSES

Published recommendations for design and code effort are not relevant to the PSP context. While cost models allow the estimation of design and code effort, small programs such as the PSP assignments do not address issues such as productizing and integration, which can each impose a three-fold increase in effort [Brooks 1995, 230]. The consequence is that there is no established "best practice" for design and coding effort in a PSP-like context. Similarly for defect density in design inspections, code inspections, and testing, the relevant studies are for software systems.

The recommended rates for inspections, however, appear relevant, although inspections are a team effort, and teams with four members are recommended [Fagan 1976, 191]. The preparation rate for code should be about 100 LOC/hour (no more than 200 LOC/hour). Rates greater than 200 LOC/hour are grounds for re-inspection.

The recommended review rate for high-level design is twice that of code, based on the estimated lines of code. The recommended review rate for detailed design is the same as that of code [Fagan 1986, 749]. Based on PSP design being analogous to detailed design and PSP review time being analogous to inspection preparation time, the unacceptable review rates are those above 200 LOC/hour for both design reviews and code reviews.

5.4 MEASURES FOR PROCESS CONTROL

The development processes to control in PSP are the design and coding processes, since there is no requirements analysis step. Defect data from the design and code reviews can be used to control design and coding. Causal analysis of signals in the defect data can point to anomalies in either the production (design or code) or review processes. Production effort and review rates are also useful measures to consider and may provide added insight into whether good engineering practice is being followed. A signal in either of these measures may correspond to a signal in the corresponding defect measures.

The measures analyzed to identify the common cause PSP system are therefore:

- *program size*, LOC
- *design time*, minutes/LOC
- *design review rate*, LOC/hour
- *defect density in design review*, defects removed in design review / KLOC
- *coding time*, minutes/LOC
- *code review rate*, LOC/hour
- *defect density in code review*, defects removed in code review / KLOC
- *defect density in compile*, defects removed in compile / KLOC
- *defect density in testing*, defects removed in testing / KLOC

5.5 TECHNIQUES FOR IDENTIFYING OUTLIERS

The natural technique for identifying atypical performance is the control chart, specifically the XmR chart for individual values and moving ranges. Although it can be argued

that control charts should not be used for PSP data since the PSP assignments do not capture a process "over time" in the sense traditionally used for process control, for a mature production process over an extended period of time, which PSP appears to be as shown in the exploration of the PSP class data in Section 4.6.3, the sequence of production is not relevant [Hahn and Meeker 1993, 6].

In recent years, SPC has been successfully applied to services as well as manufacturing [Wheeler and Poling 1998; Wheeler 2003]. The same observations that characterize the service environment – high variability in data combined from multiple individuals – also apply to using SPC on software processes. The empirical observation is that SPC is being used in both environments with success; the extension to individual data is a natural, although challenging, extension. Normally, machines operate with less variation than human teams, and human teams function with less variation than individuals. High variability does not invalidate the use of control charts.

No changes in the common cause system are expected since the PSP text and SEI-authorized instructors provide a stable instructional context. This conclusion is supported by the analysis of PSP classes in Section 4.5.3. Signals of possible assignable causes could be used by the instructors to identify individuals needing additional help, although care would be needed that a motivational use of the data did not cause dysfunctional behavior, e.g., falsified data. This suggests the view that the process being controlled is one for building PSP-trained students rather than software programs.

Assignable causes of variation should be removed when calculating control limits, but causal analysis cannot be performed when doing a retrospective analysis [Liberatore 1995, 6], therefore robust control limits are calculated using a two-stage procedure [Rocke 1988; Roes

1993]. Points outside the initial 3σ limits are removed, and robust control limits calculated. In normal SPC, points are only removed when they are confirmed by causal analysis as assignable causes of variation, but that is not feasible for a retrospective analysis. One iteration, the two-stage procedure, is a reasonable compromise for identifying the "voice of the process." The robust limits should be closer to the "true" process capability than the initial limits. Another robust technique, "robust regression," drops data points more than two standard deviations from the mean response variable, but it is only appropriate when dealing with a small number of predictor variables [Devnani-Chulani 1999, 36].

Other commonly used detection rules are run tests. These would be problematic for PSP, since the PSP as an educational process is arguably a mature, constant process. Each set of assignments comes from a different source (individual), therefore any run test would be conceptually inappropriate. The use of control charts with the PSP data is essentially an exercise in rigorously identifying outliers from a stable population.

All of the lower limits for the PSP data are less than zero, which means that the effective lower control limit is zero for the measures chosen. In some instances a data transformation might allow the potential for a lower control limit. A "low" signal for effort might indicate poor design or coding practices, which might in turn indicate a high defect injection in the design or coding phases. A "low" signal for review rate might indicate a meticulous review, perhaps suggesting concern on the part of the programmer about the quality of the design or code. A "low" signal for defect density might indicate an inadequate review, which could suggest a high escape rate. The XmR charts used in this section are unable to identify such cases.

A simple alternative to control charts for identifying outliers is to set limits at 1.5 times the interquartile range below the 25% and above the 75% quantiles [SAS Institute 2000, 36].

This interquartile limit (IQL) provides an independent check on the plausibility of the robust limits as truly representing the voice of the process.

A variety of techniques can be used for identifying outliers, including those based on the influence of data points on regression models, such as Cook's D and DFFITS [Rawlings, Pantula, and Dickey 1998, 361-364]. Care in discarding outliers is recommended, however; direct evidence of recording errors or miscalculations is desirable to justify discarding data [Neter et al. 1996, 103-104]. For a retrospective analysis, that is not practical, therefore relatively simple approaches to identifying outliers, such as XmR charts and interquartile limits, are appropriate.

5.6 IDENTIFYING SIZE OUTLIERS

The graph in Figure 14 is an *X* chart. In principle, the *mR* chart should not add insight since each data point can be considered a random pull from the PSP statistical universe, which is a variant of the argument against using run tests. Some statisticians recommend not plotting the *mR* chart in any case, arguing that the *X* chart contains all of the information available [Nelson 1982; Roes 1993].

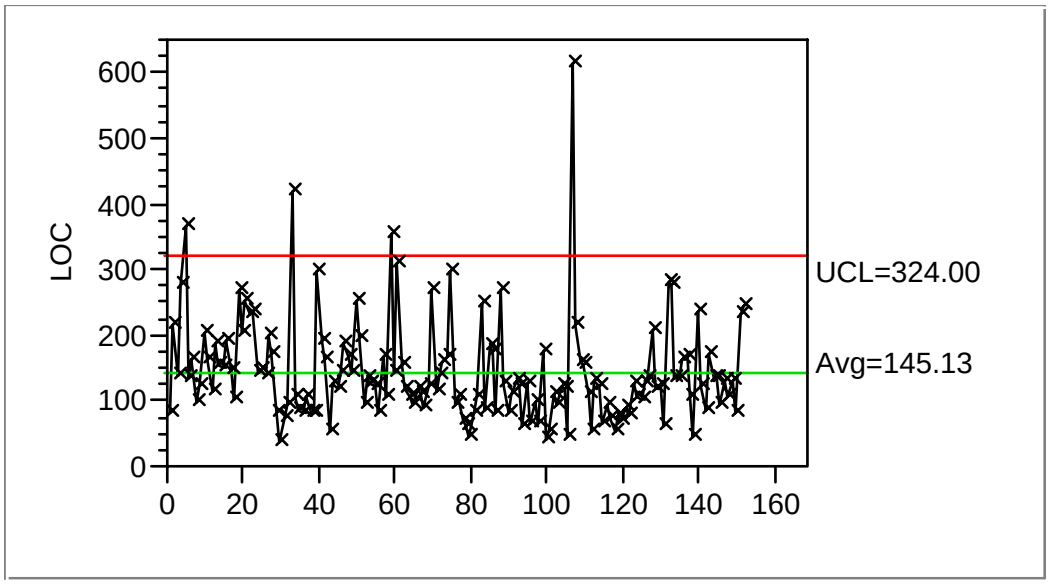

Figure 14 Initial X Chart for *Program Size* in (PSPb, C, 9A)

Size is the most commonly used factor in defect prediction models and is a critical software product measure. Atypically large programs are also atypical solutions to the PSP assignments. The above figure contains the initial control limits for *program size* for (PSPb, C, 9A). Figure 15 below contains the robust control limits, with the initial set of "outliers" removed.

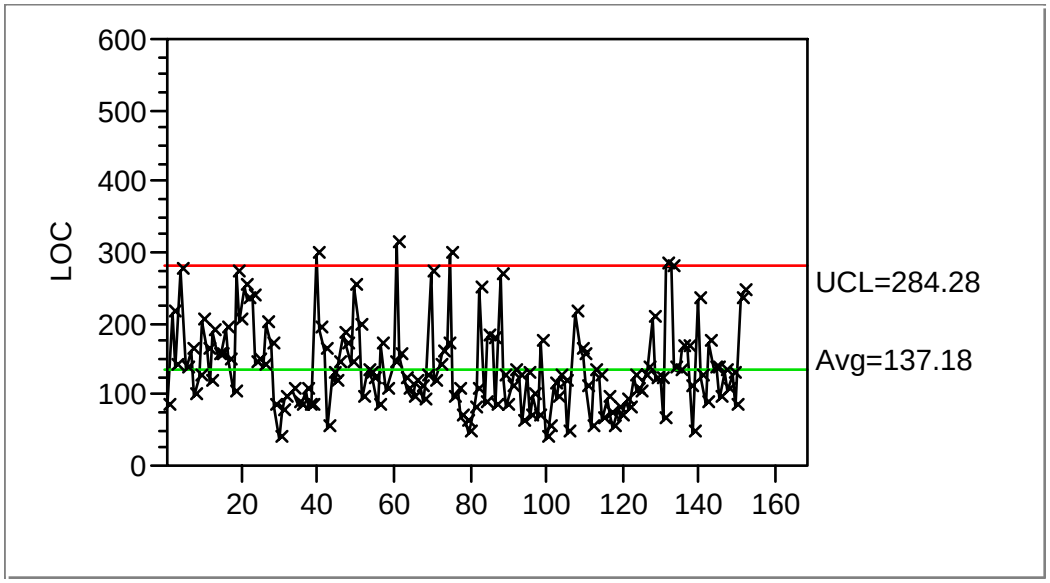

Figure 15 Robust X Chart for *Program Size* in (PSPb, C, 9A)

Table 95 contains the outlier statistics for *program size*. It shows the initial XmR, robust XmR, and interquartile limits, as well as the number of points identified as outliers for each set of limits.

Table 95 Outlier Statistics for *Program Size (LOC)*

Statistic	PSPb C 9A	PSPb C++ 9A	PSPb C 10A	PSPb C++ 10A
Initial $\overline{X}$	145.1	167.2	204.2	255.4
Initial UCL_X	324.0	424.7	430.8	613.9
Number of points outside the initial limits	4	1	4	1
Robust $\overline{X}$	137.2	159.2	187.9	245.3
Robust UCL_X	284.3	380.9	360.4	552.6
Number of points outside the robust limits	7	2	7	2
Interquartile limit	284.0	399.0	359.0	552.0
Number of points outside the IQL	7	1	7	2

An atypical size, i.e., a point be above the upper limit for *program size,* could indicate an inappropriate solution was chosen for the problem. Differing emphases on aspects of a program such as flexibility and speed can have a dramatic impact on size, which has been characterized as the "problem of ambiguous programming objectives" [Weinberg 1998, 126-132].

5.7 IDENTIFYING DESIGN OUTLIERS

5.7.1 Design Effort

Table 96 contains the outlier statistics for *design time*. It is widely believed that time spent in design has a high benefit, although the learning curve effects for the PSP design templates may counteract that effect here.

Table 96 Outlier Statistics for *Design Time*

Statistic	PSPb C 9A	PSPb C++ 9A	PSPb C 10A	PSPb C++ 10A
Initial $\overline{X}$	0.6	0.6	0.5	0.5
Initial UCL_X	1.6	1.8	1.4	1.6
Number of points outside the initial limits	2	1	4	2
Robust $\overline{X}$	0.6	0.6	0.5	0.5
Robust UCL_X	1.5	1.6	1.2	1.3
Number of points outside the robust limits	4	1	9	2
Interquartile limit	1.6	1.9	1.3	1.4
Number of points outside the IQL	2	1	9	2

Points above the upper limit for *design effort* could result from problem complexity, solution complexity, or the learning curve. Problem complexity should not be an issue since each assignment is the same for all students. Solution complexity can be a factor, especially if some students are more successful at reusing software from earlier assignments. Unfamiliarity with the application domain should not be a factor since assignments 9 and 10 are

straightforward elaborations of earlier assignments. The most likely cause for a signal in *design effort* is difficulty in learning to use the design templates.

5.7.2 Design Review Rate

Table 97 contains the outlier statistics for *design review rate.*

Table 97 Outlier Statistics for *Design Review Rate*

Statistic	PSPb C 9A	PSPb C++ 9A	PSPb C 10A	PSPb C++ 10A
Initial $\overline{X}$	525.6	746.5	588.5	537.9
Initial UCL_X	1712.9	2868.5	1907.0	1840.3
Number of points outside the initial limits	7	5	7	3
Robust $\overline{X}$	450.6	502.0	474.6	457.8
Robust UCL_X	1336.5	1637.3	1353.9	1557.1
Number of points outside the robust limits	13	10	10	5
Interquartile limit	1278.6	1473.6	1443.1	1325.0
Number of points outside the IQL	14	10	9	7

The natural process limits are well above the maximum rate of 200 LOC/hour for effective design reviews. Since the upper limit for *design review rate* is not within the specification limits, the design review process is not capable (in statistical terms), although even poor reviews are better than none at all. Since the design review process is not capable, the process improvement focus should be on ensuring that the *design review rate* conforms to recommended practice rather than analyzing signals that are outside the limits for the process.

Points above the upper limit for *design review rate* clearly indicate, in this case, that the student is not following a rigorous design review process. The comparatively large number of points outside the limits indicates that, even with an incapable process, a number of students are failing to follow the "normal design review" process as learned by most PSP students by this time in the course. Points inside the limits, but significantly higher than 200 LOC/hour, suggest that the student's behavior is normal, even if less than effective. For the 38 cases where there are signals for *design review rate*, based on the robust limits, the average *defect removal effectiveness* is 23%. Comparing this to the average *defect removal effectiveness* of 62% for design reviews with a rate less than the maximum 200 LOC/hour confirms that high review rates indicate ineffective design reviews.

5.7.3 Defect Density in Design Review

Table 98 contains the outlier statistics for *defect density in design review.*

Table 98 Outlier Statistics for *Defect Density in Design Review*

Statistic	PSPb C 9A	PSPb C++ 9A	PSPb C 10A	PSPb C++ 10A
Initial $\overline{X}$	13.5	13.5	7.3	9.7
Initial UCL_X	53.7	60.8	32.3	43.5
Number of points outside the initial limits	8	2	4	2
Robust $\overline{X}$	10.5	12.0	6.1	8.3
Robust UCL_X	45.1	53.0	26.6	37.1
Number of points outside the robust limits	15	6	7	3
Interquartile limit	50.6	46.3	29.7	31.8
Number of points outside the IQLt	10	7	7	5

Points above the upper limit for *defect density in design review* could indicate that the design review was unusually effective in identifying defects or that the student was having difficulty with the application domain.

5.8 IDENTIFYING CODING OUTLIERS

5.8.1 Coding Effort

Table 99 contains the outlier statistics for *coding time.* While high values for *design time* may indicate a deliberate approach to design, high values of *coding time* may suggest that design work is being (inappropriately) performed in coding.

Table 99 Outlier Statistics for *Coding Time*

Statistic	PSPb C 9A	PSPb C++ 9A	PSPb C 10A	PSPb C++ 10A
Initial $\overline{X}$	0.6	0.6	0.5	0.5
Initial UCL_X	1.7	1.7	1.1	1.2
Number of points outside the initial limits	4	3	6	3
Robust $\overline{X}$	0.6	0.5	0.5	0.5
Robust UCL_X	1.4	1.4	1.0	1.1
Number of points outside the robust limits	6	6	11	4
Interquartile limit	1.4	1.4	1.1	1.3
Number of points outside the IQL	6	6	10	2

Points above the upper limit for *coding effort* may suggest an inadequate design. There should be few learning curve effects associated with the coding process; by this point in PSP, any unfamiliarity with the programming language should have been largely overcome. A "low" signal for *coding effort* may indicate inadequate programming, which might in turn indicate a high defect density in the coding or testing phases, but this XmR chart is unable to identify such problems.

5.8.2 Code Review Rate

Table 100 contains the outlier statistics for *code review rate.*

Table 100 Outlier Statistics for *Code Review Rate*

Statistic	PSPb C 9A	PSPb C++ 9A	PSPb C 10A	PSPb C++ 10A
Initial $\overline{X}$	403.6	403.6	580.5	451.7
Initial UCL_X	1191.6	1239.0	1954.5	1381.8
Number of points outside the initial limits	5	2	4	1
Robust $\overline{X}$	365.7	355.9	438.7	429.4
Robust UCL_X	999.9	1024.6	1090.6	1253.1
Number of points outside the robust limits	12	4	10	2
Interquartile limit	981.7	962.3	1172.5	1031.8
Number of points outside the IQL	12	6	8	3

Similar to the case for design reviews, the average *code review rate* exceeds the maximum code inspection rate of 200 LOC/hour, and the code review process cannot be considered "capable."

Points above the upper limit for *code review rate* suggest that the student is not following a rigorous code review process, similar to the case for *design review rate*. Points inside the limits, but significantly higher than 200 LOC/hour, suggest that the student's behavior is normal, even if less than effective. For the 28 cases where there are signals for *code review rate*, based on the robust limits, the *defect removal effectiveness* is 19%. Comparing this to the average *defect removal effectiveness* of 50% for code reviews with a rate less than 200 LOC/hour confirms that high review rates indicate ineffective code reviews.

5.8.3 Defect Density in Code Review

Table 101 contains the outlier statistics for *defect density in code review*.

Table 101 Outlier Statistics for *Defect Density in Code Review*

Statistic	PSPb C 9A	PSPb C++ 9A	PSPb C 10A	PSPb C++ 10A
Initial $\overline{X}$	24.5	27.7	16.9	19.4
Initial UCL_X	90.3	97.8	58.7	66.8
Number of points outside the initial limits	6	4	2	3
Robust $\overline{X}$	21.2	22.6	15.7	16.0
Robust UCL_X	73.9	77.0	53.3	52.8
Number of points outside the robust limits	9	6	2	3
Interquartile limit	69.0	74.8	61.5	55.8
Number of points outside the IQL	12	6	2	3

Points above the upper limit for *defect density in code review* could indicate that the code review was unusually effective in identifying defects, that the design inputs were defect prone, or that the student was having difficulty with the application domain or programming language.

5.9 IDENTIFYING COMPILATION OUTLIERS

A compiler can be used as a debugging tool. One of the ongoing discussions about code inspections is whether the inspected code should come from a clean compile, i.e., the compiler

should be used to remove "obvious" syntactic defects. Table 102 contains the outlier statistics for *defect density in compilation.*

Table 102 Outlier Statistics for *Defect Density in Compilation*

Statistic	PSPb C 9A	PSPb C++ 9A	PSPb C 10A	PSPb C++ 10A
Initial $\overline{X}$	16.1	22.2	12.7	16.8
Initial UCL_X	65.8	91.1	46.5	63.2
Number of points outside the initial limits	4	3	1	1
Robust $\overline{X}$	14.0	18.8	12.4	15.9
Robust UCL_X	54.1	75.6	44.8	58.7
Number of points outside the robust limits	7	5	2	2
Interquartile limit	60.0	66.0	47.0	52.5
Number of points outside the IQL	5	6	1	4

Points above the upper limit for *defect density in compilation* could indicate that a large number of syntactic mistakes were made in coding, which might suggest a lack of expertise in the programming language. Wesslen found that PSP reviews identified a higher percentage of compile defects than design defects, suggesting that one possible improvement for PSP reviews would be a greater focus on logical errors [Wesslen 1999, 32]. It may also indicate a large number of defects escaping from earlier phases of the life cycle.

5.10 IDENTIFYING TESTING OUTLIERS

Table 103 contains the outlier statistics for *defect density in testing*, which, while of limited value as a process control measure, provides insight into the effectiveness of the other measures and indicates the stability of the overall PSP process.

Table 103 Outlier Statistics for *Defect Density in Testing*

Statistic	PSPb C 9A	PSPb C++ 9A	PSPb C 10A	PSPb C++ 10A
Initial $\overline{X}$	15.7	16.1	10.9	11.1
Initial UCL_X	56.8	60.6	39.0	43.8
Number of points outside the initial limits	6	1	2	3
Robust $\overline{X}$	13.4	14.1	10.1	9.0
Robust UCL_X	43.6	49.7	34.2	33.7
Number of points outside the robust limits	9	4	4	6
Interquartile limit	41.0	47.1	35.5	30.5
Number of points outside the IQL	9	4	4	6

Defect density in testing is not used for process control, since it is the surrogate for software quality, but high values indicate poor quality programs. A point above the upper limit for *defect density in testing* suggests that an unusual number of defects in design or coding escaped those life cycle activities and were captured in testing. Association of signals in testing with in-process signals indicates opportunities for taking corrective action earlier in development. For the 23 signals in *defect density in testing*, there is an associated in-process

signal in 15 cases. The most commonly associated signals are for *defect density in code review* and *defect density in compile* (6 and 7 instances respectively).

Unfortunately, a high percentage (33%) of the assignments have signals for one or more of the eight explanatory variables. There are 143 assignments with one or more signals for the 433 assignments. This suggests that learning occurs at the end of an assignment rather than during it, which is reasonable given the small size of the PSP assignments.

5.11 DISCUSSION OF OUTLIER IDENTIFICATION

The control charts demonstrate that signals of assignable causes may occur in the PSP data, more than would be expected by chance. One or two points outside the limits can be expected for these data sets using Wheeler's Empirical Rule that a homogenous data set will have approximately 99-100% of the data within the 3σ limits [Wheeler and Chambers 1992, 61]. Numbers greater than three or four for these PSP data sets suggest instability, i.e., some students are not consistently following the PSP process. It may also be that some students have had a bad day independent of their use of PSP… or are simply poor programmers.

The latter suggestion highlights the concern that the results of statistical analysis should not be used for motivational purposes; "grades" in PSP should depend on learning how to program better, not on performing well relative to the other students. A consistent conclusion in analyzing individual differences in programmers, however, is that investments in eliminating the lower tail of the individual differences distribution, whether via training or reassignment, provide the greatest potential benefit in improving performance [Curtis 1988, 290; Boehm 1981, 666-667]. As shown in Section 4, *disciplined processes* can contribute significantly to addressing

these issues, but effective performance, which can be supported by SPC, is needed as well as consistency.

The process instability index provides a heuristic for determining whether a process is reasonably stable [Pierce 2005]. The formula for calculating this index is:

$$S_t = (\frac{\#signals}{\#observations})*100$$

and $S_t > 3\%$ is considered statistically unstable. For the individual process variables, the processes are predominantly unstable. In three cases, one data set has $S_t < 3$, and in two other cases, two data sets have $S_t < 3$. For the other 29 data sets, $S_t > 3$.

The identification of signals by the XmR charts do not necessarily add value to the PSP process. The determination of value can be approximated by the association of in-process signals with poor software quality, if the processes are capable (the control limits are within the specification limits). As already shown, 65% of the signals in *defect density in testing* could have been identified early in the life cycle. If the full set of eight variables is used for process control, however, 33% of the assignments would undergo causal analysis at one or more points in the process – an unacceptably high rate for practical use. If only the defect data are used, one or more signals in 53 assignments are identified in development that correspond to 12 of the 23 signals in testing. Causal analysis of 12% of the assignments is more realistic, although the benefits remain limited compared to the potential.

For design and code review rates, the control limits failed to fall within recommended best practice, therefore those charts do not add value for control, and the charts simply confirm that high review rates are associated with ineffective reviews. Charts for effort provided minimal insight. Charts for defect density in reviews provided benefit, although there were a large number of false alarms.

Table 104 compares the number of outliers identified using interquartile limits to the number identified using XmR charts. A positive number indicates the number of additional signals identified by the XmR chart over the interquartile limit; a negative number indicates more signals were identified by the interquartile limits.

Table 104 Outlier Differences Between XmR Charts and Interquartile Limits

Confounding Variable	PSPb C 9A	PSPb C++ 9A	PSPb C 10A	PSPb C++ 10A
Program Size	0	1	0	0
Design Time / LOC	2	0	0	0
Design Review Rate	-1	0	1	-2
Defect Density in Design Review	5	-1	0	-2
Coding Time / LOC	0	0	1	2
Code Review Rate	0	-2	2	-1
Defect Density in Code Review	-3	0	0	0
Defect Density in Compile	2	-1	1	-2
Defect Density in Testing	0	0	0	0
Totals	**5**	**-3**	**5**	**-5**

The interquartile limits are about as effective as XmR charts in identifying outliers. Although the XmR chart identifies slightly more points than the interquartile limits, the comparison suggests that detection rules such as run tests are needed to maximize the benefits of XmR charts. Use of more sophisticated control charting techniques, such as u-charts, depends on an analysis of the distributional assumptions made by those techniques.

This analysis confirms the high variability of individual performance, even when following a disciplined process at the end of the PSP course, where the variation has decreased

relative to the *ad hoc* process at the beginning of the course. In a sense, this analysis identifies worst-case bounds for performance when following a disciplined software process. It is difficult to conceive of an industrial setting that could match the potential for individual variation found in the PSP environment, since the preferred unit of study in an industry setting is the team or project.

Even when SPC is successfully applied, there is a tension between having stable processes and continual process improvement. The process data for the previous process may not be valid for the new process, and new control limits may need to be recalculated on an ongoing basis. These process shifts are illustrated in a small way by the statistically significant differences between assignments 9 and 10 identified in Section 4.5.1, as well as the shifts between the *PSP major processes*.

5.12 CONCLUSIONS FOR OUTLIER IDENTIFICATION

One may conclude from this analysis that, although disciplined processes improve performance and decrease variation – sufficiently that the learning objectives for the PSP students are attained – the disciplined process at the end of the course remains unsatisfactory compared to the potential of conformant processes that follow generally-accepted best practice. In spite of the growing adoption of SPC (and specifically control charts) in industry, it is clear from this analysis that to get the best value from statistical techniques, a consistently-implemented process is necessary but not sufficient for statistical control. The effective implementation of recommended practices is also needed before the adjectives “disciplined” or “mature” can be used.

Many of the PSP students have not arrived at "industry best practice" with respect to review rates. The objective of PSP to induce learning based on personal data is an on-going process that is not completed within the confines of the course, as has been observed in previous studies [Hayes 1998, 65; Ferguson et al. 1997, 28-30].

As this analysis shows, if outliers are identified for an assignment for each of the eight variables that might be used for process control retrospectively, about one third of the assignments would be excluded for analysis based on one or more outliers – an excessive number to exclude without causal analysis of why the data was atypical. This does not necessarily mean that in-process control using all eight variables with causal analysis performed in real-time would not be cost effective. It does mean that a retrospective analysis of a complex process, the PSP process in this case, cannot effectively identify outliers on a per-variable basis effectively, although outlier identification per variable was useful in the simple regression models in Chapter 4. Multiple regression models, as described in Chapter 7, use regression outlier techniques to identify influential outliers more effectively.

My contributions in this analysis are therefore the following results:

- PSP processes are not statistically capable or stable by the end of the ten assignments in the PSP class.
- For a retrospective analysis, XmR control charts using only out-of-bounds signals are roughly equivalent to interquartile limits in identifying outliers. This suggests that run-based signaling techniques should also be used to identify assignable causes of variation.
- When the natural process limits for a PSP process are identified, the measured process performance may not meet recommended practice, reinforcing the need for

> continual improvement to continue after the course. The Team Software Process is the recommended mechanism for continuing professional development and deploying the PSP ideas in a team context [Humphrey 1999].

For software professionals, the implications of my research are two-fold. First, factors known to affect the review process, such as review rates, should be measured, and recommended practices should be quantified and followed. Second, data collection without analysis is not helpful for controlling performance. Measurement must be followed by analysis and action if the benefits of measurement are to be achieved. As demonstrated in the PSP data, measurement does not necessarily lead to control. Although this is intrinsic to the PSP context, which focuses on learning from measured performance rather than meeting a specific technique's requirements, i.e., the inspection rules established by Fagan [Fagan 1986], controlling and improving performance based on measurement is the ultimate objective of PSP and the related TSP and CMM work.

6.0 STATISTICAL DISTRIBUTIONS OF SOFTWARE DEFECT DATA

6.1 THE RESEARCH QUESTION: TESTING STATISTICAL DISTRIBUTIONS

The research in this chapter focuses on the statistical distributions that best describe defects in the software process. Since the choice of an appropriate statistical analysis frequently depends on assumptions about the distribution followed by the data, empirical results will help make informed decisions.

This is particularly an issue in choosing u-charts as a tool for process control since they assume a Poisson distribution. While a common choice, the empirical research on its appropriateness is mixed, and it is frequently used without checking whether the Poisson assumption is valid.

The goal of this analysis is not to argue that software defect data follow a particular distribution; the goal is to reject invalid hypotheses that the defect data follow distributions that cannot be empirically supported. Data drawn from a complex world cannot be expected to follow a mathematical formalism such as a statistical distribution exactly. The question is whether the formalism is a sufficiently accurate approximation of the real world to provide useful insight.

6.2 STATISTICS RELEVANT TO DISTRIBUTIONS

In these analyses, the distributions of concern are over modules rather than time, although much of the research previously performed, specifically for reliability models, has looked at defect detection over time. The number of defects is frequently assumed to follow a Poisson distribution; defect density is frequently assumed to follow a lognormal distribution.

The empirical distribution of a data set can be compared to a theoretical distribution such as the lognormal or Poisson distributions. For continuous distributions, such as defect density, the Shapiro-Wilk test is used to measure goodness of fit [SAS Institute 1989, 126]. For discrete distributions, such as the number of defects, the χ^2 (chi-squared) test is used to measure goodness of fit [Hogg and Ledolter 1992, 254-255]. In testing goodness-of-fit, the null hypothesis is that the empirical distribution follows a theoretical distribution; unlike other statistical analyses, it is desirable not to reject the null hypothesis. These hypotheses will be considered statistically significant at $\alpha=0.05$.

Several informal checks of distributional assumptions can be made. For the normal distribution, the mean is equal to the median, the skewness is zero, and the kurtosis is three [Wheeler 2000, 84; Leemis 1995, 60; Wheeler and Chambers 1992, 325]. Skewness measures the symmetry of a distribution (or the relative sizes of the tails); a positive skewness indicates the right-hand tail is more massive. Kurtosis measures peakedness (or the combined weight of the tails); a kurtosis value less than three is considered light-tailed. For the lognormal distribution, the informal checks are not so simple, although the log transformation of the variable follows a normal distribution; for example, the median of a lognormal distribution is at $x=e^{\mu}$ [Aitchison and Brown 1957, 8-9].

For the Poisson distribution, the mean is equal to the variance [Hogg and Ledolter 1992, 102-104]. Distributions related to the Poisson distribution include the binomial, where the variance is less than the mean, and the negative binomial, where the variance is greater than the mean [Johnson and Kotz 1969, 138]. For the exponential distribution, the mean is equal to the standard deviation, the skewness is two, and the kurtosis is nine [Leemis 1995, 85; Wheeler 2000, 84].

The data sets used in these analyses were for assignments 9 and 10 in C and C++ for PSPb. Outliers were identified using interquartile limits for the variable whose distribution is being analyzed.

6.3 DISTRIBUTION OF DESIGN DEFECTS

Table 105 summarizes statistics for the number of *defects removed in design review* both including and excluding outliers as identified by interquartile limits. The column labeled "w" includes all data points for the assignment. The column labeled "w/o" summarizes the data with the outliers for the variable removed, and the values are in bold italics to ease comparison.

Table 105 Statistics for the Number of *Defects Removed in Design Review*

Statistic	PSPb, C				PSPb, C++			
	9A		10A		9A		10A	
	with	w/o	with	w/o	with	w/o	with	w/o
Median	1	***1***	1	***1***	1	***1***	1	***1***
Mean	2.0	***1.5***	1.7	***1.1***	1.8	***1.4***	2.3	***1.6***
Interquartile Range	3	***2***	2	***2***	3	***2***	3	***2***
Standard Deviation	3.1	***1.9***	3.0	***1.6***	2.4	***1.8***	4.2	***2.0***
Variance	9.8	***3.7***	9.3	***2.5***	5.6	***3.1***	17.6	***4.1***
Skewness	3.1	***1.5***	3.8	***1.5***	2.1	***1.9***	4.8	***2.1***
Kurtosis	14.5	***1.9***	18.0	***1.6***	5.3	***4.1***	29.5	***6.0***

The variance is consistently and significantly greater than the mean for the number of *defects removed in design review*, suggesting that the Poisson distribution is a poor fit for the data (and that the negative binomial distribution might be a better fit).

Table 106 contains the results of χ^2 tests of the number of *defects removed in design review* against the negative binomial distribution. Two parameters, *p* and *k*, characterize this distribution; they are estimated using the method of moments [Williamson and Bretherton 1963, 12]. The differences between the observed frequencies and the theoretical frequencies are used to calculate q_{k-3}, which is compared to a critical value of χ^2 derived from the number of cells in the table and the specified significance level ($\alpha=0.05$). If $q_{k-3} < \chi^2(h,\alpha)$, the fit against the theoretical distribution is not rejected. None of the variables tested against the negative binomial distribution fit with outliers included, therefore all of the reported results are for data sets with outliers excluded. Statistically significant results at $\alpha=0.05$ for q_{k-3} are in bold.

Table 106 Number of *Defects Removed in Design Review* Against the Negative Binomial Excluding Outliers

Data Set	p	k	q_{k-3}	Critical Value
PSPb, C, 9A	0.405	1.021	12.844	$\chi^2(5, 0.05) = 11.070$
PSPb, C, 10A	0.440	0.864	**8.080**	$\chi^2(4, 0.05) = 9.488$
PSPb, C++, 9A	0.452	1.155	**3.007**	$\chi^2(3, 0.05) = 7.815$
PSPb, C++, 10A	0.390	1.023	**4.231**	$\chi^2(4, 0.05) = 9.488$

In three of four instances, the negative binomial distribution is not rejected as a reasonable fit to the data. In the remaining instance, a slightly more liberal choice for α would not reject the negative binomial. The negative binomial distribution therefore seems a reasonable choice for describing the number of *defects removed in design review*. Das observes that the negative binomial distribution should be considered whenever the assumption of pure randomness cannot be met in count data because of clustering [Das 2003].

Table 107 summarizes statistics for the *defect density in design review* both including and excluding outliers.

Table 107 Statistics for *Defect Density in Design Review*

Statistic	PSPb, C				PSPb, C++			
	9A		10A		9A		10A	
	with	w/o	with	w/o	with	w/o	with	w/o
Median	6.2	***5.1***	3.3	***2.2***	6.4	***5.6***	6.0	***5.0***
Mean	13.5	***9.9***	7.3	***5.5***	13.5	***9.1***	9.7	***6.9***
Interquartile Range	19.7	***16.1***	11.8	***9.9***	18.1	***12.7***	12.7	***10.0***
Standard Deviation	18.8	***12.9***	11.3	***7.0***	18.1	***11.4***	12.7	***7.5***
Variance	354.5	***166.2***	126.7	***49.5***	327.8	***129.5***	161.1	***56.3***
Skewness	1.9	***1.4***	3.0	***1.2***	1.7	***1.4***	2.2	***1.1***
Kurtosis	4.2	***1.2***	14.9	***0.3***	2.1	***1.1***	5.8	***0.7***

The hypothesis that *defect density in design review* follows either the normal or lognormal distributions is uniformly rejected by the Shapiro-Wilk test at $p\text{-value}<0.0001$ for all data sets.

6.4 DISTRIBUTION OF CODING DEFECTS

Table 108 summarizes statistics for the number of *defects removed in code review* both including and excluding outliers.

Table 108 Statistics for the Number of *Defects Removed in Code Review*

Statistic	PSPb, C				PSPb, C++			
	9A		10A		9A		10A	
	with	w/o	with	w/o	with	w/o	with	w/o
Median	2	***2***	3	***3***	3	***3***	3	***3***
Mean	3.5	***2.6***	3.4	***3.1***	4.0	***3.4***	4.7	***4.0***
Interquartile Range	3	***3***	4	***4***	5	***4***	5	***4***
Standard Deviation	4.3	***2.5***	3.8	***3.1***	3.8	***3.1***	5.7	***4.1***
Variance	18.7	***6.3***	14.4	***9.5***	14.8	***9.3***	32.2	***17.1***
Skewness	3.1	***1.8***	2.2	***1.3***	1.8	***1.2***	2.8	***2.3***
Kurtosis	13.8	***5.3***	7.1	***1.8***	4.3	***1.4***	10.0	***6.4***

The variance is consistently and significantly greater than the mean for the number of *defects removed in code review*, suggesting that the Poisson distribution is a poor fit for the data (and that the negative binomial distribution might be a better fit). Table 109 contains the results of χ^2 tests of the number of *defects removed in code review* against the negative binomial distribution.

Table 109 Number of *Defects in Code Review* Against the Negative Binomial Excluding Outliers

Data Set	p	k	q_{k-3}	Critical Value
PSPb, C, 9A	0.413	1.829	**12.648**	$\chi^2(7, 0.05) = 14.067$
PSPb, C, 10A	0.326	1.499	**19.969**	$\chi^2(13, 0.05) = 22.362$
PSPb, C++, 9A	0.366	1.963	**13.431**	$\chi^2(8, 0.05) = 15.507$
PSPb, C++, 10A	0.234	1.222	**14.474**	$\chi^2(15, 0.05) = 28.869$

In all four instances, the negative binomial distribution is not rejected as a reasonable fit to the data. The negative binomial distribution therefore seems a reasonable choice for describing the number of *defects removed in code review*.

Table 110 summarizes statistics for *defect density in code review* both including and excluding outliers.

Table 110 Statistics for *Defect Density in Code Review*

Statistic	**PSPb, C**				**PSPb, C++**			
	9A		**10A**		**9A**		**10A**	
	with	**w/o**	**with**	**w/o**	**with**	**w/o**	**with**	**w/o**
Median	19.0	***18.1***	13.8	***12.8***	19.7	***18.5***	12.6	***12.4***
Mean	24.5	***18.8***	16.9	***15.7***	27.7	***20.8***	19.4	***16.0***
Interquartile Range	24.3	***20.4***	22.5	***22.2***	25.6	***19.5***	19.3	***16.7***
Standard Deviation	25.0	***15.2***	16.5	***13.7***	30.4	***16.6***	19.9	***12.4***
Variance	623.6	***232.0***	273.8	***187.1***	924.3	***274.1***	396.4	***153.0***
Skewness	1.8	***0.7***	1.6	***0.5***	2.4	***0.9***	2.4	***0.8***
Kurtosis	3.3	***0***	5.2	***-0.9***	7.0	***0.9***	7.4	***0***

The hypothesis that *defect density in code review* follows either the normal or lognormal distributions is rejected by the Shapiro-Wilk test at $p\text{-value}<0.01$ (and at $p\text{-value}<0.0001$ for most cases). The normal distribution is rejected at $p\text{-value}=0.0004$ for (PSPb, C++, 9A, NoOutliers) and at $p\text{-value}=0.0010$ for (PSPb, C++, 10A, NoOutliers). The lognormal distribution is rejected at $p\text{-value}=0.0040$ for (PSPb, C++, 10A, Outliers).

6.5 DISTRIBUTION OF COMPILE DEFECTS

Table 111 summarizes statistics for the number of *defects removed in compile* both including and excluding outliers.

Table 111 Statistics for Number of *Defects Removed in Compile*

Statistic	**PSPb, C**				**PSPb, C++**			
	9A		**10A**		**9A**		**10A**	
	with	**w/o**	**with**	**w/o**	**with**	**w/o**	**with**	**w/o**
Median	2	***1***	2	***2***	2	***2***	3	***2.5***
Mean	2.1	***1.8***	2.6	***2.5***	3.2	***2.4***	3.9	***3.4***
Interquartile Range	3	***3***	2	***2.5***	3	***3***	5	***4***
Standard Deviation	2.6	***2.1***	3.3	***3.1***	3.8	***2.2***	4.4	***3.7***
Variance	6.9	***4.2***	10.6	***9.9***	14.5	***5.0***	19.7	***13.9***
Skewness	2.9	***2.4***	4.1	***4.4***	2.5	***1.0***	2.0	***1.9***
Kurtosis	11.2	***9.2***	27.7	***32.4***	7.9	***0.7***	5.2	***5.4***

The variance is consistently and significantly greater than the mean for the number of *defects removed in compile*, suggesting that the Poisson distribution is a poor fit for the data (and that the negative binomial distribution might be a better fit). Table 112 contains the results of χ^2 tests of the number of *defects removed in compile* against the negative binomial distribution.

Table 112 Number of *Defects Removed in Compile* Against the Negative Binomial Excluding Outliers

Data Set	p	k	q_{k-3}	Critical Value
PSPb, C, 9A	0.857	10.787	54.497	$\chi^2(6, 0.05) = 12.592$
PSPb, C, 10A	0.245	0.844	25.562	$\chi^2(11, 0.05) = 19.675$
PSPb, C++, 9A	0.480	2.215	17.083	$\chi^2(8, 0.05) = 15.507$
PSPb, C++, 10A	0.245	1.103	**9.150**	$\chi^2(9, 0.05) = 16.919$

In one instance, the negative binomial distribution is not rejected as a reasonable fit to the data. In one other instance, a slightly more liberal choice for α would not reject the negative binomial. The negative binomial distribution therefore seems worth considering for describing the number of *defects removed in compile*, but care should be taken if it is used.

Table 113 summarizes statistics for *defect density in compile* both including and excluding outliers.

Table 113 Statistics for *Defect Density in Compile*

Statistic	PSPb, C				PSPb, C++			
	9A		10A		9A		10A	
	with	w/o	with	w/o	with	w/o	with	w/o
Median	11.0	***10.6***	10.8	***10.2***	12.5	***11.8***	9.5	***8.7***
Mean	16.2	***13.7***	12.7	***12.4***	22.1	***15.6***	16.8	***13.8***
Interquartile Range	23.8	***22.7***	17.8	***18.9***	24.1	***19.4***	19.1	***17.7***
Standard Deviation	18.6	***12.8***	11.8	***11.3***	26.5	***14.9***	17.8	***13.7***
Variance	346.9	***162.8***	140.3	***128.2***	701.0	***223.5***	317.4	***186.3***
Skewness	2.4	***0.9***	0.9	***0.7***	2.0	***1.2***	1.4	***1.1***
Kurtosis	8.1	***0.6***	0.3	***-0.3***	3.8	***1.0***	1.4	***0.5***

The hypothesis that *defect density in compile* follows either the normal or lognormal distributions is uniformly rejected by the Shapiro-Wilk test at $p\text{-value}<0.0001$ for all data sets.

6.6 DISTRIBUTION OF TESTING DEFECTS

Table 114 summarizes statistics for the number of *defects removed in testing* both including and excluding outliers.

Table 114 Statistics for Number of *Defects Removed in Testing*

Statistic	PSPb, C				PSPb, C++			
	9A		10A		9A		10A	
	with	w/o	with	w/o	with	w/o	with	w/o
Median	2	***1***	2	***2***	1	***1***	2	***1***
Mean	2.1	***1.8***	1.9	***1.7***	2.2	***2.0***	2.6	***2.1***
Interquartile Range	2	***1***	2	***2***	2	***2***	2	***2***
Standard Deviation	2.3	***2.0***	1.8	***1.5***	2.4	***2.2***	3.9	***2.6***
Variance	5.3	***3.9***	3.4	***2.4***	5.6	***4.8***	15.3	***6.8***
Skewness	2.9	***3.7***	2.0	***1.3***	2.0	***2.3***	4.0	***3.1***
Kurtosis	11.7	***21.2***	6.3	***3.3***	4.6	***6.5***	20.4	***11.6***

The variance is consistently and significantly greater than the mean for the number of *defects removed in testing*, suggesting that the Poisson distribution is a poor fit for the data (and that the negative binomial distribution might be a better fit). Table 115 contains the results of χ^2 tests of the number of *defects removed in testing* against the negative binomial distribution.

Table 115 Number of *Defects Removed in Testing* Against the Negative Binomial Excluding Outliers

Data Set	p	k	q_{k-3}	Critical Value
PSPb, C, 9A	0.462	1.546	17.863	$\chi^2(4, 0.05) = 9.488$
PSPb, C, 10A	0.500	1.700	12.700	$\chi^2(6, 0.05) = 12.592$
PSPb, C++, 9A	0.417	0.431	14.216	$\chi^2(5, 0.05) = 11.070$
PSPb, C++, 10A	0.309	0.939	**12.576**	$\chi^2(3, 0.05) = 12.592$

In one instance, the negative binomial distribution is not rejected as a reasonable fit to the data. In two other instances, a slightly more liberal choice for α would not reject the negative binomial. The negative binomial distribution therefore seems worth considering for describing the number of *defects removed in testing*, but care should be taken if it is used.

Table 116 summarizes statistics for *defect density in testing* both including and excluding outliers.

Table 116 Statistics for *Defect Density in Testing*

Statistic	**PSPb, C**				**PSPb, C++**			
	9A		**10A**		**9A**		**10A**	
	with	**w/o**	**with**	**w/o**	**with**	**w/o**	**with**	**w/o**
Median	12.1	***11.8***	8.1	***7.6***	10.2	***9.8***	7.0	***6.2***
Mean	15.7	***12.7***	10.8	***9.6***	16.1	***12.6***	11.1	***7.6***
Interquartile Range	13.8	***12.7***	13.1	***12.3***	17.0	***14.5***	11.1	***8.4***
Standard Deviation	16.1	***10.4***	11.0	***8.3***	22.5	***11.1***	13.2	***6.9***
Variance	260.7	***108.2***	122.0	***68.6***	506.7	***124.2***	174.9	***47.5***
Skewness	2.0	***0.8***	2.3	***0.6***	4.9	***0.9***	2.2	***1.1***
Kurtosis	5.0	***0.3***	10.6	***-0.6***	33.4	***0***	5.8	***1.2***

The hypothesis that *defect density in testing* follows either the normal or lognormal distributions is rejected by the Shapiro-Wilk test at *p-value<0.0001* for all data sets, except (PSPb, C++, 10A, Outliers), where it is rejected at *p-value=0.0017* for the lognormal distribution.

6.7 CONCLUSIONS FOR STATISTICAL DISTRIBUTIONS

For the PSP data, my research indicates that the number of defects found in a review cannot be accurately characterized by the Poisson distribution. To the degree this conclusion can be generalized to industry projects, this is a concern since u-charts are commonly used when applying SPC to software defect data. The XmR chart, which is a robust technique in the presence of non-normal data, is a safer choice [Wheeler 2000].

If techniques using distributional assumptions are used, my results suggest that the negative binomial distribution should be considered for counts of defects. Although the results are fairly consistent for the PSP data, such assumptions should be tested against the data being analyzed in any specific case.

No good theoretical distribution was found for characterizing defect density data. Although the lognormal distribution is a plausible choice to consider, the empirical results for the PSP data indicate that distributional assumptions such as this should always be tested.

In general, when analyzing software data, the statistical techniques used should be rigorous when distributional assumptions are violated, and whatever assumptions are made should be tested against the data. The normal distribution is frequently assumed but is inappropriate for most software data. Using averages that approach a normal distribution is not feasible if there is no appropriate grouping in which averages can be calculated, whether for conceptual or pragmatic reasons.

If defect data follow a Poisson distribution, the u-chart is an appropriate technique, but if the negative binomial distribution is a better choice, control charts specifically designed for that distribution should be used [Das 2003].

Examples of robust techniques in the presence of non-normal data include the X and Xbar control charts [Schilling and Nelson 1976 ; Wheeler 2000], two-sided hypothesis tests concerning the coefficients in a linear regression, and analysis of variance for equal means [Hahn 1971, 21].

My contributions in this analysis are therefore the following results:

- Statistical assumptions, such as the distribution that data follow, should be tested where they are important for achieving correct conclusions, i.e., the statistical techniques making the assumption cannot be characterized as robust when the assumptions are violated.
- The frequently made assumption that defect data follow a Poisson distribution is not valid for the PSP defect data; a negative binomial distribution is preferable.
- Although u-charts may be commonly used in the software industry for defect data [Paulk, Goldenson, and White 2000, 58-59], their use is questionable unless the statistical assumption of a Poisson distribution has been tested.

7.0 MODELING SOFTWARE QUALITY IN PSP

7.1 THE RESEARCH QUESTION: PREDICTING DEFECTS

The research in this chapter focuses on the predictor variables for predicting software quality early in the life cycle based on process and product information. More pertinently from a management perspective, given a set of product characteristics such as *program size*, this research attempts to identify the software process factors that affect the quality of the software product.

The fundamental assumption in software process improvement, as explicitly stated in models such as the Software CMM [Paulk et al. 1995, 8], is that the software process largely determines software quality. It is, in turn, a specific instance of a fundamental assumption in TQM: good engineering and management practices (good processes) drive quality as part of a chain reaction that ultimately impacts business drivers such as cost, schedule, customer satisfaction, profitability, and market share [Deming 1986, 3]. There is an implicit assumption that competent people are doing the work. As already noted, there are dramatic differences in programmer performance even when disciplined processes are followed, and *programmer ability* is a crucial contributor to quality.

If software process factors, such as review rates, determine the quality of the software product as observed in testing, then it seems reasonable from a management perspective to encourage the good practices that result in superior quality. Even for organizations focusing on cycle time or cost, the chain reaction of quality can be expected to lessen cycle time and decrease

cost. Although the specific coefficients are likely to change in moving from the classroom to an industry environment, the factors affecting quality can be expected to remain significant.

Some factors that are not applicable in the classroom are likely to become important in industry projects, e.g., *personnel continuity*, as are other factors that are not significant for classroom-sized problems, e.g., *problem complexity*. Both kinds of factor are outside the scope of the analysis in this chapter.

7.2 DECISION POINTS IN THE PSP PROCESSES

The data for these analyses span the ten assignments in the PSP course. Processes range from *ad hoc* at the beginning to relatively disciplined in the final four assignments. Even when good practices are followed in principle, however, the implementations may not follow recommendations for best practice, e.g., reviews may not be performed at an acceptable review rate. From a research perspective, PSP provides large amounts of high-quality, detailed data that enable exploring software processes in a non-trivial way for individual professionals.

While it is clear from the analyses in Chapter 4 that the PSP process has a significant impact on software quality, the focus of these analyses is on the interactions of the underlying drivers. The primary process drivers are the design and code reviews introduced in assignment 7. There is a learning curve associated with adopting these techniques that continues after the PSP class, as demonstrated by the relatively few assignments where the recommended review rate of less than 200 LOC/hour is followed in the class and by the reports of improved performance in projects after the class [Ferguson et al. 1997; Hilburn and Humphrey 2002, 75]. These results should be considered as indicating a trend rather than the ultimate performance possible using these techniques.

Decisions made early in the life cycle tend to be more cost effective, e.g., identifying a defect-prone module in the design stage and taking corrective action versus addressing it in the testing stage. These models therefore consider the insights possible in design, in coding, and in compile as three distinct decision points in the software process.

Factors that are significant early in the life cycle may be superseded by later factors. The earlier factors are likely to drive, and be correlated with, the later factors. For example, skimping on design time may lead to increased coding and testing times as design issues are resolved later in the life cycle. These issues are explored in the analyses below.

7.3 MULTIPLE REGRESSION MODELS FOR PSP QUALITY

The multiple regression models described in this section use the general linear model. The focus in these analyses is on identifying the factors that affect software quality rather than building a specific regression model that would primarily be relevant in the PSP context.

The process variables of primary interest in predicting defects are the times spent in design and code, the review rates, and the defect densities found in the reviews. *Defect density in compile* provides data for a third decision point in addition to design and code. *Program size* and *programmer ability* address the product and people issues; they are common variables considered for all models. Note that *programmer ability* in these regression models is empirically determined by average performance on the first three assignments; with only three possible values, it is a simple surrogate for ability.

Program size is expressed in thousands-of-lines-of-code (KLOC), and times are expressed in hours. This makes measures across variables consistent. When review rates are measured in hrs/KLOC, there is a natural progression from zero (no review held) to fast reviews

to reviews that comply with the recommended review rates. The recommended review rate of less than 200 LOC/hour therefore becomes a value greater than 5 hours/KLOC.

As a rule of thumb, it is desirable to have six to ten cases in the data set per variable being analyzed [Neter et al. 1996, 330]. As a result, the data sets primarily used in these analyses are the C and C++ data sets for PSPb, which have 1758 and 920 observations respectively. This is reasonable for a rigorous analysis of models that may have as many as nine main effects and as many as 17 statistically significant interaction effects.

The variable names for the factors used in the multiple regression models in this chapter, and their definitions, are listed in Table 117.

Table 117 Variable Names and Definitions for Multiple Regression Models

Variable Name	Definition
PgmrAb	Programmer Ability (Average Defect Density in Testing 1A-3A)
KLOC	Program Size (Thousands of Lines of Code)
MajPrcs	PSP Major Process (PSP0, PSP1, PSP2, PSP3)
DDsTim	Design Time (hrs/KLOC)
EDRR	Design Review Rate (hrs/KLOC)
FDDDR	Defect Density in Design Review (defects/KLOC)
GCoTim	Coding Time (hrs/KLOC)
HCCR	Code Review Rate (hrs/KLOC)
IDDCR	Defect Density in Code Review (defects/KLOC)
JDDCm	Defect Density in Compile (defects/KLOC)
TDDTs	Defect Density in Testing (defects/KLOC)

7.3.1 An Overview of Regression Theory

The equation for the standard linear regression model in matrix form is $Y=X\beta+\xi$, where Y is the vector of observations (the dependent variable), X is the treatment design matrix for the predictor variables, β is the vector of treatment fixed-effect parameters, and ξ is the vector of experimental errors [Littell et al. 1996, 16; Neter et al. 1996, 226-230]. The β parameters are partial regression coefficients since they express the partial effect of one prediction variable when the others are held constant.

There are n observations and $p-1$ predictor variables. The errors ξ are assumed to be independent, normal variables with the expected value, $E(\xi)=0$ and $\sigma^2(\xi)=\sigma^2 I$. The least squares estimates for β are $b=(X'X)^{-1}(X'Y)$. The estimates for Y are $\hat{Y} = Xb$. The residual errors are $e = Y - \hat{Y}$.

The adjusted coefficient of multiple determination, R^2_a, may be used for comparing models. It measures the proportionate reduction of total variation in Y associated with the set of X variables. The error mean square (*MSE*) is an estimate of σ^2 and can be compared to the regression mean square (*MSR*) to test whether parameters $\beta=0$ in analysis of variance.

7.3.2 The Baseline Multiple Regression Models

The exploratory data analysis indicated that the useful variables, excluding the primary process variables, are the *PSP major process*, *program size*, and *programmer ability*. The *PSP major process* aggregates process information that will be further explored at a finer level of detail for design, coding, and compilation.

The multiple regression results for the baseline models on *defect density in testing* are shown in Table 118. The baseline regression model without interaction effects is:

$$\textit{(Defect density in testing)} = \beta_0 + \beta_{MajPrcs}\,(X_{MajPrcs}) + \beta_{PgmrAb}\,(PgmrAb) + \beta_{KLOC}\,(KLOC)$$

where $\beta_{MajPrcs}$ is the level for the *PSP major process* and $X_{MajPrcs}$ is an indicator variable for whether that *PSP major process* is the correct one for the observation. The baseline regression model with interaction effects, noted with "-IE" attached to the model name in the table headers, is:

$$\begin{aligned}\textit{(Defect density in testing)} = {} & \beta_0 + \beta_{MajPrcs}\,(X_{MajPrcs}) + \beta_{PgmrAb}\,(PgmrAb) \\ & + \beta_{KLOC}\,(KLOC) \\ & + \beta_{MajPrcs*PgmrAb}\,(X_{MajPrcs}*PgmrAb) + \beta_{MajPrcs*KLOC}\,(X_{MajPrcs}*KLOC) \\ & + \beta_{PgmrAb*KLOC}\,(PgmrAb*KLOC) + \beta_{MajPrcs*PgmrAb*KLOC}\,(X_{MajPrcs}*PgmrAb*KLOC)\end{aligned}$$

Table 118 Multiple Regression Models for the Baseline Case

Source		Baseline Model PSPb C	Baseline Model PSPb C++	Baseline Model - IE PSPb C	Baseline Model - IE PSPb C++
Model	DF	5	5	15	15
	SS	943691.3	391640.5	1430081.7	564995.4
	MS	188738.3	78328.1	95338.8	37666.4
Error	DF	1752	914	1742	904
	SS	1768047.5	804368.0	1281657.0	631013.1
	MS	1009.2	880.1	735.7	698.0
Total	DF	1757	919	1757	919
	SS	2711738.7	1196008.5	2711738.7	1196008.5
F Ratio		187.0	89.0	129.6	54.0
Prob > F		<0.0001	<0.0001	<0.0001	<0.0001
R^2_a		0.3469	0.3253	0.5255	0.4684

The baseline models were shown to be statistically significant for both of the data sets, including and excluding interaction effects. The baseline models provide a benchmark for comparing the process-based models. Any model built in design, coding, or compile that accounts for less of the variation in the software quality than the equivalent baseline model provides less insight than the relatively simple baseline model that can be used prior to beginning the software effort.

The parameter estimates of the baseline regression models, and the associated standard errors (in parentheses under the estimate), are listed in Table 119 for the main effects. Estimates where the p-values for the null hypothesis $\beta_i = 0$ are less than 0.05 are denoted with *, less than 0.01 with **, less than 0.001 with ***, and less than 0.0001 with ****.

Table 119 Main Effects for the Baseline Models

Parameter		Baseline Model PSPb C	Baseline Model PSPb C++	Baseline Model - IE PSPb C	Baseline Model - IE PSPb C++
β_0 (Intercept)		2.44 (3.71)	4.71 (4.81)	11.69 (7.37)	10.53 (19.02)
MajPrcs	**PSP0**	31.52**** (3.36)	24.36**** (4.27)	-16.57* (8.06)	-14.28 (19.39)
	PSP1	13.59**** (3.25)	9.82* (4.26)	9.08 (8.11)	5.63 (19.59)
	PSP2	-1.06 (3.30)	-2.58 (4.31)	-2.07 (8.02)	-0.10 (19.72)
	PSP3	0.0 (.)	0.0 (.)	0.0 (.)	0.0 (.)
PgmrAb		0.42**** (0.02)	0.49**** (.03)	0.08 (0.14)	0.07 (0.39)
KLOC		-62.16**** (10.86)	-58.70**** (10.32)	-16.59 (30.09)	-6.41 (75.82)

All three variables were shown to be statistically significant in the baseline models without interaction effects. When interaction effects were considered (in the last two columns of the table), they dominated the models, suggesting the complexity of the interdependencies between the ability of the programmer, the processes used by the programmer, and the size of the program being built. Note that terms including categorical variables such as the *PSP major process* are not uniquely estimable and are estimated relative to one of the categories, e.g., PSP3, which has an estimate of 0 and no standard error.

The interaction effects for the *PSP major process* are separately listed in Table 120.

Table 120 Interaction Effects for the *PSP Major Process* in the Baseline Models

Parameter		Baseline Model PSPb C	Baseline Model PSPb C++	Baseline Model - IE PSPb C	Baseline Model - IE PSPb C++
MajPrcs * PgmrAb	**PSP0**	--	--	1.25**** (0.15)	1.40*** (0.40)
	PSP1	--	--	0.27 (0.15)	0.39 (0.40)
	PSP2	--	--	0.11 (0.15)	0.23 (0.40)
	PSP3	--	--	0.0 (.)	0.0 (.)
MajPrcs * KLOC	**PSP0**	--	--	147.89** (46.59)	121.43 (79.34)
	PSP1	--	--	9.98 (41.51)	-27.11 (81.58)
	PSP2	--	--	14.50 (38.72)	-18.72 (83.74)
	PSP3	--	--	0.0 (.)	0.0 (.)
MajPrcs * PgmrAb * KLOC	**PSP0**	--	--	-6.34**** (0.98)	-6.73*** (1.77)
	PSP1	--	--	-1.26 (0.81)	-0.62 (1.76)
	PSP2	--	--	-0.42 (0.74)	-0.73 (1.77)
	PSP3	--	--	0.0 (.)	0.0 (.)

The difference between PSP0 and the other PSP processes is clearly dominant, indicating that any disciplined process is superior to an *ad hoc* process.

The interaction effects for *programmer ability* are separately listed in Table 121.

Table 121 Interaction Effects for *Programmer Ability* in the Baseline Models

Parameter		Baseline Model PSPb C	Baseline Model PSPb C++	Baseline Model - IE PSPb C	Baseline Model - IE PSPb C++
MajPrcs * PgmrAb	**PSP0**	--	--	1.25**** (0.15)	1.40*** (0.40)
	PSP1	--	--	0.27 (0.15)	0.39 (0.40)
	PSP2	--	--	0.11 (0.15)	0.23 (0.40)
	PSP3	--	--	0.0 (.)	0.0 (.)
PgmrAb * KLOC		--	--	-0.16 (0.61)	-0.07 (1.62)
MajPrcs * PgmrAb * KLOC	**PSP0**	--	--	-6.34**** (0.98)	-6.73*** (1.77)
	PSP1	--	--	-1.26 (0.81)	-0.62 (1.76)
	PSP2	--	--	-0.42 (0.74)	-0.73 (1.77)
	PSP3	--	--	0.0 (.)	0.0 (.)

Programmer ability appears to be most important as a factor when an *ad hoc* process is followed. This is consistent with the findings in Chapter 4 that a disciplined process decreases the differences between the top-quartile and bottom-quartile performers in PSP.

The interaction effects for *program size* are included in the separate tables for interaction effects of the other two variables. *Program size* appears to only be statistically significant in conjunction with PSP0 when interaction effects are considered.

All three variables in the baseline models can be considered useful predictor variables for software quality.

7.3.3 Multiple Regression Models in Design

In the process-based models, the *PSP major process* variable is superseded by the variables characterizing the process in greater depth. For the design process, those variables are the *design time*, *design review rate*, and *defect density in design review*. The multiple regression results for the design models are shown in Table 122. The design regression model without interaction effects (named Design or Design Additive hereafter) is:

$$\textit{(Defect density in testing)} = \beta_0 + \beta_{PgmrAb}\,(PgmrAb) + \beta_{KLOC}\,(KLOC) + \beta_{DDsTim}\,(DDsTim) + \beta_{EDRR}\,(EDRR) + \beta_{FDDDR}\,(FDDDR)$$

The design regression model with interaction effects (named Design –IE or Design Additive –IE hereafter) is:

$$\textit{(Defect density in testing)} = \beta_0 + \beta_{PgmrAb}\,(PgmrAb) + \beta_{KLOC}\,(KLOC) + \beta_{DDsTim}\,(DDsTim) + \beta_{EDRR}\,(EDRR) + \beta_{FDDDR}\,(FDDDR) + \beta_{PgmrAb*KLOC}\,(PgmrAb*KLOC) + \beta_{PgmrAb*DDsTim}\,(PgmrAb*DDsTim) + \beta_{PgmrAb*EDRR}\,(PgmrAb*EDRR) + \beta_{PgmrAb*FDDDR}\,(PgmrAb*FDDDR)$$

$+ \beta_{DDsTim*FDDDR}\ (DDsTim*FDDDR) + \beta_{EDRR*FDDDR}\ (EDRR*FDDDR)$

$+ \beta_{PgmrAb*KLOC*EDRR}\ (PgmrAb*KLOC*EDRR)$

$+ \beta_{PgmrAb*KLOC*FDDDR}\ (PgmrAb*KLOC*FDDDR)$

$+ \beta_{PgmrAb*DDsTim*EDRR}\ (PgmrAb*DDsTim*EDRR)$

$+ \beta_{PgmrAb*EDRR*FDDDR}\ (PgmrAb*EDRR*FDDDR)$

$+ \beta_{KLOC*DDsTim*EDRR}\ (KLOC*DDsTim*EDRR)$

$+ \beta_{DDsTim*EDRR*FDDDR}\ (DDsTim*EDRR*FDDDR)$

$+ \beta_{PgmrAb*DDsTim*EDRR*FDDDR}\ (PgmrAb*DDsTim*EDRR*FDDDR)$

$+ \beta_{PgmrAb*KLOC*DDsTim*EDRR*FDDDR}\ (PgmrAb*KLOC*DDsTim*EDRR*FDDDR)$

Table 122 Multiple Regression Models in Design

Source		Design Model PSPb C	Design Model PSPb C++	Design Model - IE PSPb C	Design Model - IE PSPb C++
Model	DF	5	5	19	19
	SS	735278.5	328348.6	1022548.6	496651.9
	MS	147055.7	65669.7	53818.3	26139.6
Error	DF	1752	914	1738	900
	SS	1976460.2	867659.9	1689190.2	699356.5
	MS	1128.1	949.3	971.9	777.1
Total	DF	1757	919	1757	919
	SS	2711738.7	1196008.5	2711738.7	1196008.5
F Ratio		130.4	69.2	55.37	33.64
Prob > F		<0.0001	<0.0001	<0.0001	<0.0001
R^2_a		0.2691	0.2706	0.3703	0.4029

The design models were shown to be statistically significant for both of the data sets, including and excluding interaction effects. Perhaps the most important observation about the design models, however, is that they not account for as much of the variation in the data as the baseline models. Since the primary difference between the two sets of models is the replacement of the *PSP major process* variable with the three design process variables, it seems reasonable to infer that other aspects of the process, which are holistically captured by the simple *PSP major process*, are necessary to understand the relationship of process and software quality, at least within the PSP context.

The parameter estimates of the design models, and the associated standard errors (in parentheses under the estimate), are listed in Table 123 for the main effects.

Table 123 Main Effects for the Design Models

Parameter	Design Model PSPb C	Design Model PSPb C++	Design Model - IE PSPb C	Design Model - IE PSPb C++
β_0 (Intercept)	21.21**** (1.86)	15.37**** (2.45)	5.72* (2.48)	-0.20 (3.34)
PgmrAb	0.42**** (0.02)	0.47**** (0.03)	0.83**** (0.04)	0.95**** (0.06)
KLOC	-95.43**** (10.62)	-62.82**** (10.27)	63.99**** (16.70)	57.31*** (17.07)
DDsTim	0.31*** (0.09)	0.49**** (0.11)	0.27 (0.17)	0.17 (0.19)
EDRR	-2.48**** (0.35)	-3.28**** (0.52)	0.39 (0.81)	1.54 (1.14)
FDDDR	-0.06 (0.09)	0.33*** (0.09)	-0.52 (0.33)	-0.44 (0.47)

All three design variables, along with *programmer ability* and *program size*, were shown to be statistically significant in at least one of the design models without interaction effects. When interaction effects were considered, the design variables were no longer shown to be statistically significant, suggesting the complexity of the interdependencies between the ability of the programmer, the design processes used by the programmer, and the size of the program being built. All three design variables in the design models can be considered useful predictor variables for software quality.

All interactions were investigated, but only the 14 interaction effects that were shown to be statistically significant for at least one of the data sets were retained. The focus of this analysis is on understanding the importance of the predictor variables, rather than building a parsimonious defect prediction model. This approach was also used for the coding and compilation models. The two-factor interaction effects are separately listed in Table 124.

Table 124 Two-Factor Interaction-Effect Estimates in the Design Models

Parameter	Design Model PSPb C	Design Model PSPb C++	Design Model - IE PSPb C	Design Model - IE PSPb C++
PgmrAb*KLOC	--	--	-4.01**** (0.34)	-3.40**** (0.42)
PgmrAb*DDsTim	--	--	-0.002 (0.002)	0.003 (0.002)
PgmrAb*EDRR	--	--	-0.18**** (0.02)	-0.15**** (0.03)
PgmrAb*FDDDR	--	--	-0.02**** (0.004)	-0.01* (0.007)
DDsTim*FDDDR	--	--	0.05** (0.02)	0.04 (0.02)
EDRR*FDDDR	--	--	-0.004 (0.07)	0.04 (0.09)

Programmer ability is involved in interaction effects with every other variable in the design models, although the interaction with *design time* was not shown to be statistically significant when all statistically significant interaction effects for either data set were joined.

The possibility of an interaction between *design review rate* and *defect density in design review* was raised in Section 4.8.3, but the interaction effect was not shown to be statistically significant when all statistically significant interaction effects for either data set were joined (and the signs of the coefficient differ). Although *design review rate* was shown to be correlated with *defect removal effectiveness*, it seems likely that *defect density in design review* is also significantly affected by the number of defects in the design, and this relationship supersedes that with *design review rate*.

The interaction effects involving more than two factors are separately listed in Table 125. Interaction effects involving more than two factors are usually of little practical impact but are included in the models for completeness.

Table 125 Other Interaction-Effect Estimates in the Design Models

Parameter	Design Model PSPb C	Design Model PSPb C++	Design Model -IE PSPb C	Design Model -IE PSPb C++
PgmrAb*KLOC *EDRR	--	--	0.57**** (0.10)	0.27 (0.14)
PgmrAb*KLOC *FDDDR	--	--	0.06* (0.03)	0.04 (0.03)
PgmrAb *DDsTim*EDRR	--	--	0.006**** (0.0006)	0.004**** (0.0007)
PgmrAb*EDRR *FDDDR	--	--	0.006**** (0.001)	0.002 (0.001)
KLOC*DDsTim *EDRR	--	--	-2.02**** (0.43)	-1.19* (0.55)
DDsTim*EDRR *FDDDR	--	--	-0.002 (0.003)	-0.0007 (0.004)
PgmrAb*DDsTim *EDRR*FDDDR	--	--	-0.0002**** (0.00005)	-0.00009 (0.00008)
PgmrAb*KLOC *DDsTim*EDRR *FDDDR	--	--	-0.0001 (0.0002)	0.0002 (0.0003)

The interaction effect for *programmer ability* with *design time*, and with *design review rate*, was shown to be statistically significant for both data sets. *Programmer ability* was shown to be a statistically significant factor as a main effect and as an interaction effect with every other

variable in the design models; it is arguably the most important factor in understanding software quality during design.

As a defect prediction model, the design models are inadequate in comparison with the baseline models. From a management perspective, the design models reinforce the importance of *programmer ability* in building high quality programs.

7.3.4 Multiple Regression Models in Coding

The variables added to the code models are the *coding time*, *code review rate*, and *defect density in code review*. The multiple regression results for the code models are shown in Table 126. The code model without interaction effects (named Code or Code Additive hereafter) is:

$$\begin{aligned}(\text{Defect density in testing}) = {} & \beta_0 + \beta_{PgmrAb}(PgmrAb) + \beta_{KLOC}(KLOC) \\ & + \beta_{DDsTim}(DDsTim) + \beta_{EDRR}(EDRR) + \beta_{FDDDR}(FDDDR) \\ & + \beta_{GCoTim}(GCoTim) + \beta_{HCRR}(HCRR) + \beta_{IDDCR}(IDDCR)\end{aligned}$$

The code model with interaction effects, (named Code –IE or Code Additive –IE hereafter) is:

$$\begin{aligned}(\text{Defect density in testing}) = {} & \beta_0 + \beta_{PgmrAb}(PgmrAb) + \beta_{KLOC}(KLOC) \\ & + \beta_{DDsTim}(DDsTim) + \beta_{EDRR}(EDRR) + \beta_{FDDDR}(FDDDR) \\ & + \beta_{GCoTim}(GCoTim) + \beta_{HCRR}(HCRR) + \beta_{IDDCR}(IDDCR) \\ & + \beta_{PgmrAb*KLOC}(PgmrAb*KLOC) + \beta_{PgmrAb*EDRR}(PgmrAb*EDRR) \\ & + \beta_{PgmrAb*FDDDR}(PgmrAb*FDDDR) + \beta_{PgmrAb*GCoTim}(PgmrAb*GCoTim) \\ & + \beta_{PgmrAb*HCRR}(PgmrAb*HCRR) + \beta_{PgmrAb*IDDCR}(PgmrAb*IDDCR) \\ & + \beta_{KLOC*HCRR}(KLOC*HCRR) + \beta_{EDRR*GCoTim}(EDRR*GCoTim) \\ & + \beta_{EDRR*HCRR}(EDRR*HCRR) + \beta_{HCRR*IDDCR}(HCRR*IDDCR) \\ & + \beta_{PgmrAb*KLOC*GCoTim}(PgmrAb*KLOC*GCoTim)\end{aligned}$$

$+ \beta_{PgmrAb*KLOC*HCRR}$ *(PgmrAb*KLOC*HCRR)*

$+ \beta_{PgmrAb*DDsTim*EDRR}$ *(PgmrAb*DDsTim*EDRR)*

$+ \beta_{PgmrAb*EDRR*HCRR}$ *(PgmrAb*EDRR*HCRR)*

$+ \beta_{PgmrAb*FDDDR*IDDCR}$ *(PgmrAb*FDDDR*IDDCR)*

$+ \beta_{PgmrAb*KLOC*GCoTim*HCRR*IDDCR}$

*(PgmrAb*KLOC*GCoTim*HCRR*IDDCR)*

Table 126 Multiple Regression Models in Code

Source		**Code Model PSPb C**	**Code Model PSPb C++**	**Code Model -IE PSPb C**	**Code Model -IE PSPb C++**
Model	DF	8	8	25	25
	SS	893503.7	419377.4	1221840.6	598696.3
	MS	111688.0	52422.2	48873.6	23947.9
Error	DF	1749	911	1732	894
	SS	1818235.0	776631.1	1489898.2	597312.1
	MS	1039.6	852.5	860.2	668.1
Total	DF	1757	919	1757	919
	SS	2711738.7	1196008.5	2711738.7	1196008.5
F Ratio		107.4	61.5	56.8	35.8
Prob > F		<0.0001	<0.0001	<0.0001	<0.0001
R^2_a		0.3264	0.3449	0.4426	0.4866

The code models were shown to be statistically significant for both of the data sets, including and excluding interaction effects. The code models account for more of the variation in the data than the baseline models only for the C++ data sets. Detailed process data for the

design and code processes provides, at best, marginally superior models to the models based on the *PSP major process*.

The parameter estimates of the code models, and the associated standard errors (in parentheses under the estimate), are listed in Table 127 for the main effects.

Table 127 Main Effects for the Code Models

Parameter	Code Model PSPb C	Code Model PSPb C++	Code Model -IE PSPb C	Code Model -IE PSPb C++
β_0 (Intercept)	13.81**** (2.03)	2.35 (2.65)	5.04 (2.94)	5.05 (4.14)
PgmrAb	0.41**** (0.02)	0.41**** (0.03)	0.76**** (0.04)	0.55**** (0.08)
KLOC	-73.14**** (10.47)	-36.75*** (10.06)	72.36**** (18.27)	24.59 (17.69)
DDsTim	0.15 (0.09)	0.05 (0.11)	0.09 (0.09)	-0.14 (0.11)
EDRR	-0.45 (0.42)	-2.40*** (0.70)	1.01 (1.33)	-2.32 (1.81)
FDDDR	0.14 (0.08)	0.33*** (0.09)	-0.03 (0.14)	0.44* (0.21)
GCoTim	0.59**** (0.06)	1.21**** (0.12)	0.06 (0.12)	0.28 (0.18)
HCRR	-2.06**** (0.48)	-0.34 (0.67)	1.33 (1.11)	2.90 (1.50)
IDDCR	-0.16** (0.05)	0.02 (0.07)	-0.40*** (0.11)	-0.13 (0.15)

All three code variables were shown to be statistically significant in at least one of the code models. *Design time* dropped out of all the code models as a statistically significant main effect, suggesting that, at least within the PSP context, *design time* is superseded by *code time*, which is reasonable since the PSP assignments are primarily programming problems.

All interactions were investigated, but only the 17 interaction effects that were shown to be statistically significant for at least one of the data sets were retained. The two-factor interaction effects are separately listed in Table 128.

Table 128 Two-Factor Interaction-Effect Estimates in the Code Models

Parameter	Code Model PSPb C	Code Model PSPb C++	Code Model -IE PSPb C	Code Model -IE PSPb C++
PgmrAb*KLOC	--	--	-3.65**** (0.44)	-0.91 (0.49)
PgmrAb*EDRR	--	--	-0.05** (0.02)	-0.08* (0.03)
PgmrAb*FDDDR	--	--	-0.0003 (0.003)	-0.007 (0.004)
PgmrAb*GCoTim	--	--	0.006**** (0.001)	0.02**** (0.002)
PgmrAb*HCRR	--	--	-0.13**** (0.02)	-0.06* (0.03)
PgmrAb*IDDCR	--	--	0.002 (0.001)	-0.001 (0.003)
KLOC*HCRR	--	--	-21.28** (7.07)	-8.80 (7.72)
EDRR*GCoTim	--	--	0.06 (0.03)	0.10* (0.05)
EDRR*HCRR	--	--	-0.26 (0.15)	-0.42* (0.18)
HCRR*IDDCR	--	--	0.06**** (0.01)	0.02 (0.01)

Although the possibility of an interaction between *design time* and *code time* was raised in Sections 4.8.1 and 4.8.3, no statistically significant interaction was shown. The original concern was that an increase in *design time* corresponded to an increase in *design defect density*, possibly as a result of design activities occurring in coding; the loss of statistical significance for

design time weighs against this potential explanation. No interaction was shown to be statistically significant for *design time* with *defect density in design review* or *defect density in code review*, which lessens the likelihood that bad design decisions are being addressed in reviews as defects are identified.

The possibility that reuse of defect-free components is lessening *design time* can be tentatively ruled out since the interaction effect between *design time* and *program size* was not shown to be statistically significant. This suggests that reuse is not a major factor in decreasing design effort.

This leaves the possibility that the relatively simple PSP assignments do not require powerful design techniques, with the implication that learning curve effects associated with the PSP design techniques are driving the positive relation of *design time* to *defect density in testing*. This suggests that students taking more time to understand the PSP design techniques are also more prone to defects. It remains possible that design activities are occurring in coding, but they are relatively minor. There is, however, no direct way of testing whether this explanation has empirical support given the PSP data available.

The interaction effects involving more than two factors are separately listed in Table 129.

Table 129 Other Interaction-Effect Estimates in the Code Models

Parameter	Code Model PSPb C	Code Model PSPb C++	Code Model -IE PSPb C	Code Model -IE PSPb C++
PgmrAb*KLOC *GCoTim	--	--	-0.03 (0.02)	-0.15**** (0.03)
PgmrAb*KLOC *HCRR	--	--	0.80**** (0.12)	0.24 (0.16)
PgmrAb *DDsTim*EDRR	--	--	-0.0002 (0.0005)	0.002**** (0.0006)
PgmrAb*EDRR *HCRR	--	--	0.009**** (0.002)	0.008** (0.003)
PgmrAb*FDDDR *IDDCR	--	--	0.00007 (0.00004)	-0.000005 (0.00002)
PgmrAb*KLOC *DDsTim*EDRR *FDDDR	--	--	0.00001 (0.0001)	0.0003 (0.0002)
PgmrAb*KLOC *GCoTim*HCRR *IDDCR	--	--	-0.0002* (0.00007)	-0.00006 (0.00008)

While it is interesting to note that *programmer ability* and *program size* interact with the three design variables and with the three code variables as five-factor interaction effects, the lack of statistical significance and the small size of the coefficients suggest that these two effects are of little practical import.

As a defect prediction model, the code models are on the cusp of becoming preferable to the baseline models. From a management perspective, the code models again reinforce the importance of *programmer ability*.

7.3.5 Multiple Regression Models in Compile

The variable added to the compile models is *defect density in compile*. The multiple regression results for the compile models are shown in Table 130. The compile model without interaction effects (named Compile or Compile Additive hereafter) is:

$$\begin{aligned}(\textit{Defect density in testing}) = {} & \beta_0 + \beta_{PgmrAb}\,(PgmrAb) + \beta_{KLOC}\,(KLOC) \\ & + \beta_{DDsTim}\,(DDsTim) + \beta_{EDRR}\,(EDRR) + \beta_{FDDDR}\,(FDDDR) \\ & + \beta_{GCoTim}\,(GCoTim) + \beta_{HCRR}\,(HCRR) + \beta_{IDDCR}\,(IDDCR) \\ & + \beta_{JDDCm}\,(JDDCm)\end{aligned}$$

The compile model with interaction effects (named Compile –IE or Compile Additive –IE hereafter) is:

$$\begin{aligned}(\textit{Defect density in testing}) = {} & \beta_0 + \beta_{PgmrAb}\,(PgmrAb) + \beta_{KLOC}\,(KLOC) \\ & + \beta_{DDsTim}\,(DDsTim) + \beta_{EDRR}\,(EDRR) + \beta_{FDDDR}\,(FDDDR) \\ & + \beta_{GCoTim}\,(GCoTim) + \beta_{HCRR}\,(HCRR) + \beta_{IDDCR}\,(IDDCR) \\ & + \beta_{JDDCm}\,(JDDCm) \\ & + \beta_{PgmrAb*KLOC}\,(PgmrAb*KLOC) + \beta_{PgmrAb*GCoTim}\,(PgmrAb*GCoTim) \\ & + \beta_{PgmrAb*HCRR}\,(PgmrAb*HCRR) + \beta_{PgmrAb*IDDCR}\,(PgmrAb*IDDCR) \\ & + \beta_{PgmrAb*JDDCm}\,(PgmrAb*JDDCm) + \beta_{KLOC*GCoTim}\,(KLOC*GCoTim) \\ & + \beta_{KLOC*HCRR}\,(KLOC*HCRR) + \beta_{KLOC*JDDCm}\,(KLOC*JDDCm) \\ & + \beta_{GCoTim*JDDCm}\,(GCoTim*JDDCm) + \beta_{HCRR*IDDCR}\,(HCRR*IDDCR) \\ & + \beta_{PgmrAb*KLOC*GCoTim}\,(PgmrAb*KLOC*GCoTim) \\ & + \beta_{PgmrAb*KLOC*IDDCR}\,(PgmrAb*KLOC*IDDCR) \\ & + \beta_{PgmrAb*KLOC*JDDCm}\,(PgmrAb*KLOC*JDDCm)\end{aligned}$$

$+ \beta_{PgmrAb*FDDDR*IDDCR}$ *(PgmrAb*FDDDR*IDDCR)*

$+ \beta_{FDDDR*IDDCR*JDDCm}$ *(FDDDR*IDDCR*JDDCm)*

$+ \beta_{PgmrAb*KLOC*GCoTim*JDDCm}$*(PgmrAb*KLOC*GCoTim*JDDCm)*

$+ \beta_{PgmrAb*FDDDR*IDDCR*JDDCm}$*(PgmrAb*FDDDR*IDDCR*JDDCm)*

Table 130 Multiple Regression Models in Compile

Source		**Compile Model PSPb C**	**Compile Model PSPb C++**	**Compile Model -IE PSPb C**	**Compile Model -IE PSPb C++**
Model	DF	9	9	26	26
	SS	1002755.9	471233.4	1270016.2	682000.2
	MS	111417.3	52359.3	48846.8	26230.8
Error	DF	1748	910	1731	893
	SS	1708982.9	724775.1	1441722.6	514008.3
	MS	977.7	796.5	832.9	575.6
Total	DF	1757	919	1757	919
	SS	2711738.7	1196008.5	2711738.7	1196008.5
F Ratio		114.0	65.7	58.7	45.6
Prob > F		<0.0001	<0.0001	<0.0001	<0.0001
R^2_a		0.3665	0.3880	0.4604	0.5577

The compile models were shown to be statistically significant for both of the data sets, including and excluding interaction effects. The compile models account for more of the variation in the data than the baseline models for three of the four data sets. The compile models are somewhat superior to the baseline models and provide the basis for further exploration of the detailed process data.

The parameter estimates of the compile models, and the associated standard errors (in parentheses under the estimate), are listed in Table 131 for the main effects.

Table 131 Main Effects for the Compile Models

Parameter	Compile Model PSPb C	Compile Model PSPb C++	Compile Model -IE PSPb C	Compile Model -IE PSPb C++
β_0 (Intercept)	9.01**** (2.02)	-3.53 (2.66)	7.97* (3.16)	20.98**** (4.52)
PgmrAb	0.37**** (0.02)	0.36**** (0.03)	0.48**** (0.05)	0.006 (0.09)
KLOC	-56.27**** (10.28)	-22.36* (9.89)	-1.39 (25.38)	-53.16* (25.58)
DDsTim	0.10 (0.09)	0.04 (0.11)	-0.008 (0.08)	0.10 (0.10)
EDRR	-0.14 (0.41)	-1.90** (0.68)	0.23 (0.40)	-1.20* (0.59)
FDDDR	0.12 (0.08)	0.29** (0.09)	-0.03 (0.12)	0.06 (0.10)
GCoTim	0.37**** (0.07)	0.97**** (0.12)	-0.18 (0.16)	0.48 (0.27)
HCRR	-1.55*** (0.47)	0.16 (0.65)	-1.51 (0.85)	-0.65 (1.05)
IDDCR	-0.14** (0.05)	0.005 (0.07)	-0.17 (0.09)	-0.10 (0.13)
JDDCm	0.16**** (0.02)	0.17**** (0.02)	0.13** (0.04)	-0.34**** (0.06)

Defect density in compile was shown to be statistically significant in all four of the compile models. It is interesting to note that, in general, all of the other process variables were shown not to be statistically significant when interaction effects were included in the compile models. This reinforces the importance of the interactions between *programmer ability*, *program size*, and the process in determining software quality.

From a management perspective, the message appears clear that the competence of the staff and the processes they follow are synergistic in achieving high-quality software products. While this message is hardly surprising, in the sometimes passionate discussions of process discipline and professional heroics [Bach 1994], one side of the equation may be emphasized to the detriment of the other.

All interactions were investigated, but only the 17 interaction effects that were shown to be statistically significant for either or both of the data sets were retained. The two-factor interaction effects are separately listed in Table 132.

Table 132 Two-Factor Interaction-Effect Estimates in the Compile Models

Parameter	Compile Model PSPb C	Compile Model PSPb C++	Compile Model -IE PSPb C	Compile Model -IE PSPb C++
PgmrAb*KLOC	--	--	-1.15* (0.50)	0.65 (0.55)
PgmrAb*GCoTim	--	--	0.01**** (0.002)	0.003 (0.004)
PgmrAb*HCRR	--	--	-0.03*** (0.009)	0.01 (0.02)
PgmrAb*IDDCR	--	--	-0.005**** (0.001)	-0.001 (0.003)
PgmrAb*JDDCm	--	--	0.0007 (0.0005)	0.009**** (0.0009)
KLOC*GCoTim	--	--	3.17 (2.12)	-2.08 (2.65)
KLOC*HCRR	--	--	0.63 (4.54)	-3.40 (5.86)
KLOC*JDDCm	--	--	0.64 (0.58)	2.37**** (0.47)
GCoTim*JDDCm	--	--	-0.003**** (0.0007)	0.002 (0.001)
HCRR*IDDCR	--	--	0.05**** (0.009)	0.02** (0.007)

The interaction effects involving more than two factors are separately listed in Table 133.

Table 133 Other Interaction-Effect Estimates in the Compile Models

Parameter	Compile Model PSPb C	Compile Model PSPb C++	Compile Model -IE PSPb C	Compile Model -IE PSPb C++
PgmrAb*KLOC *GCoTim	--	--	-0.13**** (0.03)	0.02 (0.05)
PgmrAb*KLOC *IDDCR	--	--	0.02*** (0.007)	-0.003 (0.01)
PgmrAb*KLOC *JDDCm	--	--	-0.02 (0.009)	-0.04**** (0.007)
PgmrAb*FDDDR *IDDCR	--	--	0.00003 (0.00003)	-0.00009** (0.00003)
FDDDR*IDDCR *JDDCm	--	--	-0.00001 (0.00008)	0.00006 (0.00004)
PgmrAb*KLOC *GCoTim *JDDCm	--	--	0.0004 (0.0004)	-0.0006 (0.0003)
PgmrAb*FDDDR *IDDCR*JDDCm	--	--	0.000001 (0.0000009)	0.0000005 (0.0000008)

As a defect prediction model, the compile models are preferable to the baseline models. The compile models are the focus of the remaining analyses in this chapter, and the emphasis will be on regression diagnostics and on increasing the amount of variation explained by the compile models from the 37-56% measured by the adjusted coefficient of determination.

From a management perspective, all of the process-based models emphasize the importance of *programmer ability*. This is not a surprising result. Neither is the fact that, with the possible exception of *design time*, the process variables are useful predictor variables for software quality. The lack of significance for *design time* may be driven by the simple nature of

the PSP assignments compared to the complex design needs in a real-world software project; evidence for alternate explanations is lacking in the statistical results.

7.3.6 Multicollinearity and Variance Inflation Factors

Multicollinearity occurs when the predictor variables in the regression equation are correlated. When there is significant multicollinearity, the regression coefficients cannot be interpreted as reflecting the effects of the different predictor variables; the regression coefficients depend on which other predictor variables are included in the model and which ones are left out [Neter et al. 1996, 285-295].

The variance inflation factors (*VIF*) for the predictor variables can be used to detect the presence of multcollinearity. If the maximum *VIF* value is greater than ten, multicollinearity is likely to be a problem, and if the mean *VIF* values are considerably larger than one, it indicates serious multicollinearity issues [Neter et al. 1996, 386-387]. Table 134 lists the maximum and average VIF values for the compile models.

Table 134 Multicollinearity Diagnostics Using VIF

Multicollinearity Diagnostics	Compile Model PSPb C	Compile Model PSPb C++
Maximum VIF	3.0	5.0
Average VIF	1.7	2.3

Multicollinearity is a concern for these data sets, but arguably not a serious problem.

7.3.7 Influential Outliers

A number of standard diagnostics are available for identifying outliers that may be influential in the context of a regression model: leverage, the studentized deleted residual,

Cook's distance, and DFFITS. Leverage addresses the question of whether the predictor variable X_{ij} (or the set of X_{ij} values) is atypical. The studentized deleted residual addresses the question of whether the dependent variable Y_i is atypical. Cook's distance and DFFITS address the question of whether omission of the observation would produce a dramatic change in the parameter estimates for the regression model.

Leverage, h_{ii}, which measures the distance from the i^{th} case to the center of all X observations, can be used to identify potential outliers in the predictor variables. If $h_{ii} > 2\frac{p}{n}$, where p is the number of variables in the regression equation, and n is the number of observations, then the observation is a potential outlier. In this case, the number of observations is quite large for each of the data sets: (PSPb, C) has 1758 observations, and (PSPb, C++) has 920 observations. As a result, this rule is not practical for these data sets. An alternative rule is to identify an observation as a potential outlier if $h_{ii} > 0.5$, which was the rule used here [Neter et al. 1996, 375-378].

The studentized deleted residual, t_i, can be used to identify outliers for the predictor variable. The Bonferroni test, $t_i > t(1-\frac{\alpha}{2n}; n-p-1)$, is used to identify outliers [Neter et al. 1996, 373-375]. For these data sets, $t(0.9995,\infty)=3.291$, was used.

Cook's distance, D_i, measures the effect of an observation on $\hat{B}$ [Neter et al. 1996, 380-382]. If $D_i > F(0.50,p,n-p)$, then the fitted values obtained with and without the i^{th} observation can be considered to differ substantially. For these data sets, $D_i > F(0.50,27,\infty) = 1.03$ was used to identify influential observations.

$DFFITS_i$ measures the effect of an observation on $\hat{Y}_i$. If $DFFITS_i > 2\sqrt{\frac{p}{n}}$, then the ith observation is influential [Neter et al. 1996, 378-380]. For (PSPb, C), the test is for $DFFITS_i > 0.151$ for the main-effects compile model and for $DFFITS_i > 0.248$ for the compile model with interactions; for (PSPb, C++), the test is for $DFFITS_i > 0.209$ for the main-effects compile model and for $DFFITS_i > 0.343$ for the compile model with interactions.

Table 135 summarizes the results for the compile models, including and excluding influential outliers. Any observation that violated one or more of the heuristics was considered an influential outlier.

Table 135 Comparing Compile Models Including and Excluding Influential Outliers

Model and Data Set	Outliers	N Outliers	R^2_a	*MSE*	*SSE*
Compile model PSPb, C	with	47	0.3665	977.7	1708982.9
	w/o	--	0.3380	513.1	872722.4
Compile model PSPb, C++	with	28	0.3880	796.5	724775.1
	w/o	--	0.2929	461.6	388579.1
Compile model –IE PSPb, C	with	53	0.4604	832.9	1441722.6
	w/o	--	0.4038	440.6	774573.0
Compile model –IE PSPb, C++	with	36	0.5577	575.6	514008.3
	w/o	--	0.4475	393.1	336902.0

The influential outliers were subsets of the outliers identified by both the XmR charts and the interquartile limits in Chapter 5. The R^2_a values for the models without influential outliers were not as good as those for the original models. The mean sum of squares of the errors (*MSE)* for the models, which is an estimate of σ^2, decreased by 32-48%, however, and decreasing variability is desirable. Minimizing the sum of squares of the errors (*SSE*) is also desirable.

As described in Section 5.11, the analysis of outliers for the eight process variables using XmR charts and interquartile limits led to 33% of the assignments being identified as potential outliers. Although in-process control would be preferred for identifying potential assignable causes (and is the proper use of XmR charts), causal analysis is not feasible for a retrospective analysis, and excluding 33% of the data is impractical. Shewhart chose 3σ limits for control charts in part because of the theoretical maximum of 11% of the data being outside the 3σ limits; the 3σ limits were considered economically reasonable [Shewhart 1939, 91]. Wheeler's Empirical Rule suggests that it is most common for 99-100% of the data to be within 3σ of the average [Wheeler and Chambers 1992, 61]. Identifying 2-4% of the data as atypical, as the regression diagnostics do, provides a more practical set of outliers in this context than identifying outliers for each process variable independently.

7.3.8 Box-Cox Transformations

As already shown in Chapter 6, the PSP data is not described well by theoretical distributions such as lognormal or exponential. Box-Cox power transformations of the dependent variable attempt to provide a simple, normal linear model that simultaneously satisfies constant variance, normality, and $E(\hat{Y}) = X\beta$ [Rawlings, Pantula, and Dickey 1998, 409-412]. A power transformation of $\lambda=0.3$ was found to be the best value for both (PSPb, C) and (PSPb, C++). None of the models using the Box Cox transformation were superior to those using non-transformed values in terms of R^2_a, and the sum of squares of the error (*SSE*), which is minimized as the criterion for choosing the best transformation, is better for the models with non-transformed variables.

7.3.9 Multiplicative Models

A multiplicative model, with log transformations of the continuous variables, would appear to be a natural model of the software process since a chain reaction of interactions from injecting defects in production to detecting and removing defects in reviews can be expected across the life cycle. We expect defect density in design reviews and code reviews to correspond to *defect density in testing*, as was demonstrated in Sections 4.8.3 and 4.8.6. We expect review rates to affect *defect removal effectiveness*, as was demonstrated in Sections 4.8.2 and 4.8.5. We expect *design time* to affect *design defect density*, and *coding time* to affect *code defect density*, and that defects early in the life cycle will result in an increase in defects at the end of the development. This was demonstrated in Sections 4.8.1 and 4.8.4, even though an increase in time corresponded to an increase in defect density in both cases. A variety of statistically significant interaction effects in the multiple regression models in this chapter suggest complex relationships between these factors.

Table 136 contains a comparison of the additive compile models already investigated with their multiplicative counterparts. All of the models were shown to be statistically significant at $\alpha=0.01$. The multiplicative compile model without interaction effects (named Compile Multiplicative hereafter) is:

$$Ln(\text{Defect density in testing}) = \beta_0 + \beta_{PgmrAb}\,[Ln(PgmrAb)] + \beta_{KLOC}\,[Ln(KLOC)] + \beta_{DDsTim}\,[Ln(DDsTim)] + \beta_{EDRR}\,[Ln(EDRR)] + \beta_{FDDDR}\,[Ln(FDDDR)] + \beta_{GCoTim}\,[Ln(GCoTim)] + \beta_{HCRR}\,[Ln(HCRR)] + \beta_{IDDCR}\,[Ln(IDDCR)] + \beta_{JDDCm}\,[Ln(JDDCm)]$$

The multiplicative compile model with interaction effects (named Compile Multiplicative –IE hereafter) is:

$$Ln(\text{Defect density in testing}) = \beta_0 + \beta_{PgmrAb}\,[Ln(PgmrAb)] + \beta_{KLOC}\,[Ln(KLOC)]$$

$$+ \beta_{DDsTim}\,[Ln(DDsTim)] + \beta_{EDRR}\,[Ln(EDRR)] + \beta_{FDDDR}\,[Ln(FDDDR)]$$

$$+ \beta_{GCoTim}\,[Ln(GCoTim)] + \beta_{HCRR}\,[Ln(HCRR)] + \beta_{IDDCR}\,[Ln(IDDCR)]$$

$$+ \beta_{JDDCm}\,[Ln(JDDCm)]$$

$$+ \beta_{PgmrAb*KLOC}\,[Ln(PgmrAb)*Ln(KLOC)]$$

$$+ \beta_{PgmrAb)*Ln(GCoTim}\,[Ln(PgmrAb)*Ln(GCoTim)]$$

$$+ \beta_{PgmrAb*HCRR}\,[Ln(PgmrAb)*Ln(HCRR)]$$

$$+ \beta_{PgmrAb*IDDCR}\,[Ln(PgmrAb)*Ln(IDDCR)]$$

$$+ \beta_{PgmrAb*JDDCm}\,[Ln(PgmrAb)*Ln(JDDCm)]$$

$$+ \beta_{KLOC*GCoTim}\,[Ln(KLOC)*Ln(GCoTim)]$$

$$+ \beta_{KLOC*HCRR}\,[Ln(KLOC)*Ln(HCRR)]$$

$$+ \beta_{KLOC*JDDCm}\,[Ln(KLOC)*Ln(JDDCm)]$$

$$+ \beta_{GCoTim*JDDCm}\,[Ln(GCoTim)*Ln(JDDCm)]$$

$$+ \beta_{HCRR*IDDCR}\,[Ln(HCRR)*Ln(IDDCR)]$$

$$+ \beta_{PgmrAb*KLOC*GCoTim}\,[Ln(PgmrAb)*Ln(KLOC)*Ln(GCoTim)]$$

$$+ \beta_{PgmrAb*KLOC*IDDCR}\,[Ln(PgmrAb)*Ln(KLOC)*Ln(IDDCR)]$$

$$+ \beta_{PgmrAb*KLOC*JDDCm}\,[Ln(PgmrAb)*Ln(KLOC)*Ln(JDDCm)]$$

$$+ \beta_{PgmrAb*FDDDR*IDDCR}\,[Ln(PgmrAb)*Ln(FDDDR)*Ln(IDDCR)]$$

$$+ \beta_{FDDDR*IDDCR*JDDCm}\,[Ln(FDDDR)*Ln(IDDCR)*Ln(JDDCm)]$$

$$+ \beta_{PgmrAb*KLOC*GCoTim*JDDCm}[Ln(PgmrAb)*Ln(KLOC)*Ln(GCoTim)*Ln(JDDCm)]$$

$$+ \beta_{PgmrAb*FDDDR*IDDCR*JDDCm}[Ln(PgmrAb)*Ln(FDDDR)*Ln(IDDCR)*Ln(JDDCm)]$$

Table 136 Comparing Additive and Multiplicative Compile Models

Data Set	Compile Model	Outliers	R^2_a	*MSE*	*SSE*
PSPb C	Additive	with	0.3665	977.7	1708982.9
		w/o	0.3380	513.1	872722.4
	Multiplicative	with	0.3266	1039.3	1816675.9
		w/o	0.3589	496.9	845196.2
PSPb C++	Additive	with	0.3880	796.5	724775.1
		w/o	0.2929	461.6	388579.1
	Multiplicative	with	0.3156	890.7	810506.1
		w/o	0.3041	433.5	382386.2
PSPb C	Additive -IE	with	0.4604	832.9	1441722.6
		w/o	0.4038	440.6	774573.0
	Multiplicative -IE	with	0.4224	891.4	1543008.0
		w/o	0.4296	441.7	741111.6
PSPb C++	Additive -IE	with	0.5577	575.6	514008.3
		w/o	0.4475	393.1	336902.0
	Multiplicative -IE	with	0.5144	631.9	564309.3
		w/o	0.4463	394.0	337625.6

In general, the comparison between additive and multiplicative models suggests:

- the additive models including influential outliers account for more of the variation than the other models
- the multiplicative models excluding influential outliers account for more of the variation than the additive models excluding influential outliers
- *MSE* and *SSE* decrease markedly for all models when influential outliers are excluded

The multiplicative compile models are not noticeably superior to the additive models for the PSP data sets. This is not an expected result, but some researchers have found linear models preferred over non-linear models for defect prediction models that are functions of program structural characteristics and elapsed implementation time [Nikora 1998, 138].

7.3.10 Stratifying by Programmer Quartile

Programmer ability is related to all aspects of the software process, even when characterized by the simple three-category *programmer quartiles*. Since top-quartile performers have less variability, as well as better performance, models using data from only the top performers may be superior to the inclusive models already analyzed.

Table 137 compares the multiple regression models for the top-quartile performers in additive and multiplicative forms, including and excluding outliers, to the additive models for all the PSP students including outliers. All of the models were shown to be statistically significant at $\alpha=0.01$. The additive compile models are the same as those described in Section 7.3.5; the multiplicative compile models are the same as those described in Section 7.3.9. Only the data sets differ.

Table 137 Comparing Compile Models for Top-Quartile Students

Data Set	Compile Model	Outliers	R^2_a	*MSE*	*SSE*
PSPb C	Additive	with	0.3665	977.7	1708982.9
PSPb C TQ	Additive	with	0.1227	195.5	91316.8
		w/o	0.2068	96.3	40151.3
	Multiplicative	with	0.1695	1.7	798.7
		w/o	0.2725	1.4	590.4
PSPb C++	Additive	with	0.3880	796.5	724775.1
PSPb C++ TQ	Additive	with	0.0830	220.2	59016.9
		w/o	0.1847	90.5	22163.9
	Multiplicative	with	0.1038	1.7	447.3
		w/o	0.1933	1.4	336.2
PSPb C	Additive -IE	with	0.4604	832.9	1441722.6
PSPb C TQ	Additive -IE	with	0.1520	189.0	85051.9
		w/o	0.2633	113.1	45451.9
	Multiplicative -IE	with	0.1959	1.7	745.1
		w/o	0.2815	1.5	584.9
PSPb C++	Additive -IE	with	0.5577	575.6	514008.3
PSPb C++ TQ	Additive -IE	with	0.2110	189.5	47556.2
		w/o	0.2404	91.3	20443.6
	Multiplicative -IE	with	0.2393	1.4	355.6
		w/o	0.2321	1.3	295.6

In general, the comparison between the additive and multiplicative models for the top-quartile students suggests:

- the additive models for all of the PSP students, including influential outliers, account for more of the variation than the models for just the top-quartile students
- *MSE* and *SSE* decrease markedly for the models for the top-quartile students, especially when influential outliers are excluded
- the multiplicative models for the top-quartile students account for more of the variation than the additive models for the top-quartile students

The shift from additive to multiplicative models when the data are restricted to the top-quartile students fits the mental model of the process interactions better, but the stratification also leads to a less useful model in terms of accounting for variation. The decrease in *MSE*, the estimator for σ^2, is also desirable, even if perhaps not as desirable as an increase in R^2_a.

7.3.11 Stratifying by Conformant Processes

The common wisdom is that programmers who conform to recommended practice will have better results than those who do not. The performance of programmers following recommended practice, i.e., review rates less than 200 LOC/hour, should therefore be better and more predictable than that of the population as a whole. Table 138 summarizes the results of analyzing processes that conform to recommended practice for review rates. All of the models were shown to be statistically significant at $\alpha=0.01$. The additive compile models are the same as those described in Section 7.3.5; the multiplicative compile models are the same as those described in Section 7.3.9. Only the data sets differ.

Table 138 Comparing Compile Models for Conformant Processes

Data Set	Compile Model	Outliers	R^2_a	*MSE*	*SSE*
PSPb C	Additive	with	0.3665	977.7	1708982.9
PSPb C Conformant	Additive	with	0.5337	461.3	39670.3
		w/o	0.2503	167.1	13032.4
	Multiplicative	with	0.1664	1.8	157.4
		w/o	-0.0784	1.7	132.8

The first thing to note is that there were insufficient observations with reviews that satisfy recommended practice to generate a useful multiple regression model for the C++ data sets and for the models that have interaction effects. The rule of thumb is that there should be six to ten cases for every potential predictor variable [Neter et al. 1996, 330], which means there should be at least 54 observations for the main-effects compile models and 156 for the compile models with interaction effects.

This is a consequence of the PSP being a learning environment. While students are aware of the rules for inspections, the purpose of the personal software process is to identify the most effective review process for the individual programmer. While it is expected that most students should conform to recommended practice, part of the learning experience is to experientially and quantitatively go through that learning process, therefore most students do not meet the review rate initially (and many may continue to converge to "their best personal review rate" after the PSP class is over if they continue to use the PSP's measurement-driven improvement approach for personal learning).

The additive model for the conformant process, including influential outliers, accounts for more of the variation than any other model. When influential outliers are excluded, the value of R^2_a decreases, which is not desired, but the value of *MSE* also decreases, which is desirable.

One way of addressing the inadequate number of observations is to not split the data sets by *programming language* used. Table 139 summarizes the results of analyzing processes that conform to recommended practice for review rates for PSPa and PSPb. All of the models were shown to be statistically significant at $\alpha=0.01$.

Table 139 Comparing Compile Models for Conformant Processes Without Considering Programming Language

Data Set	Compile Model	Outliers	R^2_a	*MSE*	*SSE*
PSPa Conformant	Additive	with	0.1777	505.2	43948.4
		w/o	0.2131	347.6	29198.3
	Multiplicative	with	0.0683	2.3	202.6
		w/o	0.0449	2.3	194.9
PSPb C Conformant	Additive	with	0.5337	461.3	39670.3
PSPb Conformant	Additive	with	0.3239	1418.2	707674.2
		w/o	0.2858	501.4	245191.7
	Multiplicative	with	0.1631	2.1	1061.9
		w/o	0.1511	2.0	992.9
PSPb Conformant	Additive –IE	with	0.4372	1180.4	568966.7
		w/o	0.2736	451.5	212209.9
	Multiplicative –IE	with	0.1630	2.1	1025.9
		w/o	0.1365	2.0	961.9

There were insufficient observations with reviews that satisfy recommended practice to generate a useful multiple regression model for the PSPa data set with interaction effects. The additive model for the conformant process, including influential outliers, continues to account for more of the variation than any other model.

In general, the comparison between the additive and multiplicative models for the conformant processes suggests:

- *MSE* and *SSE* decrease markedly for the conformant processes, especially when influential outliers are excluded
- the additive models for the conformant processes account for more of the variation than the multiplicative ones

The lack of observations with conformant reviews is a concern since PSP is intended to instill good engineering behaviors in its students. As has already been discussed, however, PSP is also intended to be a learning experience where measured performance drives improvement actions. The lack of conformant reviews reinforces the conclusion that following recommended practice is not easy, and quantified benefits are helpful in reinforcing preferred behaviors.

7.3.12 Discussion of the Multiple Regression Models

Processes that conform to recommended practice for reviews appear to be superior to those that do not, but the scarcity of observations for the conformant processes within the various data splits make that conclusion less compelling than it could be. From a management perspective, however, the importance of quality control and quality assurance are reinforced in achieving high-quality products.

Removing influential outliers reduces the variance markedly, although it also reduces the amount of variation explained by the models. The majority of the outliers are associated with nonconformant design or code reviews. From a management perspective, this implies that identifying atypical instances of the process, for example, by applying statistical process control techniques, would improve the predictability of performance.

The additive models are, in general, superior to their multiplicative variants with the exception of data sets restricted to the data from the top-quartile performers. The additive models are therefore preferred, at least within the context of general linear models (multiplicative models are preferred for the more sophisticated mixed models in Section 7.4).

Models with interaction effects are, not surprisingly, superior to those that do not include interaction effects.

The preferred multiple regression model is the additive compile model with interactions and applied to software processes that conform to recommended practice. Influential outliers may be included to maximize R^2_a or excluded to minimize *MSE*. The normality of the residuals for the additive compile regression model with interactions for conformant processes and excluding influential outliers was checked using the Shapiro-Wilk test. The null hypothesis that residuals follow a normal distribution could not be rejected for (PSPb, C) with *p-value=0.1282* or for (PSPb, C++) with *p-value=0.3739*.

The variability explained by the multiple regression models may be more than the baseline models by compile time, but the simple baseline model with interactions is close enough in performance early enough in the life cycle that it would be preferred over the design and code models.

Published software defect prediction models accounting for 60-70% of the variability in the data with only a small number of predictor variables, perhaps only *program size*, are common. In some cases this may be because of the small data sets used; several researchers used the Akiyama data set, which has only seven data points [Akiyama 1972; Halstead 1977, 88-91; Gaffney 1984], and studies based on classroom data with fewer than 30 data points are common

[Wohlin and Petersson 2001, 343; Takahasi and Kamayach 1985, 330]. In other cases, it seems likely that team effects reduced the variation.

It is clear that *programmer ability* is a crucial variable. It is statistically significant as a main effect and in interaction with multiple other variables, even though the surrogate used is not a sophisticated one. It is also clear that a disciplined process, specifically one that conforms to good programming practices, materially improves the quality of the software product.

Interaction effects between the process variables, *programmer ability*, and *program size* are significant factors. A more sophisticated statistical analysis that addresses the significant differences between individuals is called for. The assumption of independence has been deliberately ignored in these regression models since multiple observations from each student are included in the data sets. This issue is addressed in the context of repeated measures in the mixed models in Section 7.4.

From a management perspective, emphasis on both the competence of the staff and the processes they follow is crucial for achieving high-quality software products. Overly emphasizing either process or people factors, to the detriment of the other, is counterproductive.

7.4 MIXED MODELS FOR PSP QUALITY

In analyzing the PSP data, natural growth curves can be used to model the relationships between various explanatory variables and software quality since an increase in performance is expected for students across the course. The repeated measures ANOVA used in earlier analyses of the PSP data [Hayes and Over 1997; Wesslen 2000] is a special case of growth curve models that focuses only on the factor means (the means of the treatments) [Duncan et al. 1999, 13]. Growth curve models preserve the concept that individual differences are both meaningful and

important, even when everyone develops the same way [Duncan et al. 1999, 3]. Mixed models can be used for analyzing growth curves using repeated data.

7.4.1 An Overview of Mixed Model Theory

In statistical analysis, blocks are formed that are as nearly homogeneous as possible, and treatments (or explanatory factors) are randomly assigned to blocks. Many commonly used statistical tools, such as the general linear model focus on fixed effects, i.e., those where the treatments in the analysis are the only ones for which inferences are to be made.

Random effects are those where the blocks in the analysis are a subset of a larger set of blocks in the population for which the researcher wishes to make inferences. Mixed models address both fixed and random effects, and the general linear mixed model equation is:

$$Y = X\beta + Z u + e$$

where

Y is the vector of observations

X is the matrix of values of the predictor variables (treatment design matrix)

β is the vector of regression parameters (treatment fixed effect parameters)

Z is the block design matrix

u is the vector of random block effects, assumed to be multivariate normal, $MVN(0, G)$

e is the vector of errors, assumed to be $MVN(0,R)$

G and *R* are covariance matrices that are required to be positive definite [Littell et al. 1996, 491-493]. These assumptions – that positive definite covariance matrices *G* and *R* can be constructed from the random and repeated measures variables – are satisfied in all of the models described in this chapter, although workarounds were necessary in some instances and are described at the appropriate points. The covariance matrix for *Y* is:

$$V = Z G Z' + R$$

where *Y* is assumed to be $MVN(X\beta, ZGZ'+R)$. $X\beta$ is defined by the fixed effects specified in the the mixed model.

For the PSP data, the repeated measures are for the students across the ten assignments. The *Y* vector contains the *defect density in testing* observations. Rather than being a separate variable, the ability of the programmers is incorporated into the mixed models as repeated measures that capture improving quality across the PSP assignments. Repeated measures analysis addresses the independence issue noted earlier, i.e., that data from the same student may be analyzed for multiple assignments. *Program size* and the process variables are the fixed effects contained in *X*. The confounding variables, such as academic degrees and experience, which were explored in Section 4.6, can be more rigorously explored as random effects captured in the *Z* matrix.

The mixed model is an appropriate tool for analyzing models containing both fixed and random effects and is theoretically superior to tools that focus on fixed effects. The general linear model, for example, essentially treats random effects as fixed effects. In the case of PSP, student data is being used to make inferences about the larger population of programmers, therefore random effects need to be appropriately addressed.

There are several reasons why a mixed model is preferred over a fixed effects model [Littell et al. 1996]:

- 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 programmers 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.
- In linear model theory, a linear function of (fixed) model effects is estimable if it can be written as a linear combination of expected values of the observations. Estimable functions do not depend on random effects. Linear combinations of fixed and random effects are called predictable functions. If all model effects are considered fixed, correct estimable functions in mixed models where the coefficients for all random effects are zero are falsely declared nonestimable.
- Specific random effects, or linear functions of random effects, can be estimated using best linear unbiased predictors (BLUP), which are unique to mixed model theory. Fixed effect models use best linear unbiased estimates (BLUE). To the degree that

random effects such as *years of experience* or *programming language* are significant in the mixed models, estimates incorporating all effects, whether fixed or random, correctly 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 statistical tests is likely to be low. Unfortunately the PSP data for the random effects to be explored was not consistently captured, so missing data are an issue that needs to be appropriately addressed.

The mixed model deals with repeated measures, where multiple measurements of a response variable on the same subject are taken [Littell et al. 1996, 87-134; Khattree and Naik 1999, 247-301]. 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 is unrealistic since observations close in time are likely to be more highly correlated than those far apart in time.

The covariance structure for the repeated measures is specified by the analyst, and a number of options are available [Littell et al. 1996, 269-274]. In the unstructured (UN) case, no mathematical pattern is imposed on the covariance structure. This is the most general structure. In the compound symmetry (CS) structure, variances are homogenous, and correlation is

constant. For AR(1), the autoregressive order 1 structure, variances are homogeneous, and correlations decline exponentially with distance. The UN, CS, and AR(1) covariance structures are the most commonly used; these three and five others were explored for the PSP data. Typically a model fit criterion, such as Akaike's Information Criterion (AIC), is used to select the most appropriate covariance structure and model (smaller values of AIC are better) [Littell et al. 1996, 101-102; Khattree and Naik 1999, 264], although a likelihood ratio test is preferred for rigorous conclusions [Khattree and Naik 1999, 255-265].

For the mixed models reported below, the unstructured covariance structure is used. It has the smallest *AIC* value of all the covariance structures investigated. A likelihood ratio test was performed for each covariance structure [Khattree and Naik 1999, 255-265], and the unstructured covariance structure was consistently better than the other structures at $\alpha=0.005$.

When investigating the mixed models, a multiplicative model was found to reduce the *AIC* by 2-3 times compared to the additive model in all cases. As a result, the models studied for the mixed models are all multiplicative models. This result suggests that multiplicative effect of the process variables, which is conceptually correct but not visible in the multiple regression models, is better addressed in the mixed models.

7.4.2 Mixed Models in Design

The dimensions and fit statistics for the design mixed models for *defect density in testing* are shown in Table 140, where the X matrix of fixed effects contains the *program size* and the design process variables, the repeated measures are for the students across assignments, and there are no random effects. Influential outliers are included.

There are no random effects in this model. The repeated measures across assignments are captured in the *R* matrix for the error terms, which is a 10x10 matrix. The design mixed model without interaction effects (named Design Mixed hereafter) for the fixed effects is:

$$Ln(\text{Defect density in testing}) = \beta_0 + \beta_{KLOC}\,[Ln(KLOC)] + \beta_{DDsTim}\,[Ln(DDsTim)] + \beta_{EDRR}\,[Ln(EDRR)] + \beta_{FDDDR}\,[Ln(FDDDR)]$$

No statistically significant interaction effects were found for the design mixed models, so there is no design mixed model with interaction effects.

Table 140 Mixed Models for Design

	Design Mixed PSPb C	Design Mixed PSPb C++
Covariance Parameters	55	55
Columns in X	5	5
Columns in Z	0	0
Subjects	197	108
Observations Used	1758	920
Prob > ChiSq	<0.0001	<0.0001
AIC	5888.3	3103.0

The parameter estimates of the fixed main effects for the design mixed models, and the associated standard errors, are listed in Table 141 for the data sets including influential outliers. In the following tables, for the null hypotheses $H_0: \beta_i=0,$ a *p-value*<0.05 is indicated with *, a *p-value*<0.01 is indicated with **, a *p-value*<0.001 is indicated with ***, and a *p-value*<0.0001 is indicated with ****.

Table 141 Fixed-Effect Estimates for the Design Mixed Models

Parameter	Design Mixed PSPb C (std err)	Design Mixed PSPb C++ (std err)
Intercept	1.51**** (0.15)	1.45**** (0.17)
KLnKLOC	-0.17*** (0.05)	-0.30**** (0.06)
DLnDsTim	0.22**** (0.04)	0.08 (0.06)
ELnDRR	-0.36**** (0.03)	-0.30**** (0.04)
FLnDDDR	0.13*** (0.04)	0.09 (0.05)

As *program size* increases, *defect density in testing* decreases, which is consistent with the findings of some other researchers [Basili and Perricone 1984; Shen et al. 1985]. There is some evidence that as *design time* increases, *defect density in testing* increases, which is consistent with the findings in Section 4.8.1. As the *design review rate* increases in terms of hours/KLOC, *defect density in testing* decreases. Given this formulation of review rate, the result is consistent with the belief that following recommended practice will result in higher quality software. There is some evidence that as *defect density in design review* increases, *defect density in testing* also increases, which is also as expected.

The dimensions and fit statistics for the design mixed models for *defect density in testing* are shown in Table 142 for the mixed models excluding influential outliers.

Table 142 Mixed Models for Design Excluding Outliers

	Design Mixed PSPb C	Design Mixed PSPb C++
Covariance Parameters	55	55
Columns in X	5	5
Columns in Z	0	0
Subjects	197	108
Observations Used	1711	892
Prob > ChiSq	<0.0001	<0.0001
AIC	5687.2	2981.7

The results for the design mixed models excluding outliers are consistent with those for the design mixed models including outliers. The design mixed models excluding outliers had a consistently lower *AIC* than the models including outliers.

The parameter estimates of the fixed main effects for the design mixed models, and the associated standard errors, are listed in Table 143 for the data sets excluding outliers.

Table 143 Fixed-Effect Estimates for the Design Mixed Models Excluding Outliers

Parameter	Design Mixed PSPb C (std err)	Design Mixed PSPb C++ (std err)
Intercept	1.80**** (0.15)	1.65**** (0.17)
KLnKLOC	-0.07 (0.05)	-0.17* (0.07)
DLnDsTim	0.20**** (0.05)	0.09 (0.07)
ELnDRR	-0.36**** (0.03)	-0.33**** (0.04)
FLnDDDR	0.12** (0.04)	0.10 (0.05)

The parameter estimates of the fixed main effects for the design mixed models excluding outliers are consistent with those of the design mixed models including outliers.

7.4.3 Mixed Models in Coding

The dimensions and fit statistics for the code mixed models for *defect density in testing* are shown in Table 144, where the X matrix of fixed effects contains the *program size* and the design and code process variables, the repeated measures are for the students across assignments, and there are no random effects. Influential outliers are included.

There are no random effects in this model. The repeated measures across assignments are captured in the *R* matrix for the error terms, which is contained in the SAS output in Appendix C, along with the values for the unstructured covariance matrix. The code mixed model without interaction effects (named Code Mixed hereafter) for the fixed effects is:

$$Ln(\text{Defect density in testing}) = \beta_0 + \beta_{KLOC}\,[Ln(KLOC)]$$
$$+ \beta_{DDsTim}\,[Ln(DDsTim)] + \beta_{EDRR}\,[Ln(EDRR)] + \beta_{FDDDR}\,[Ln(FDDDR)]$$
$$+ \beta_{GCoTim}\,[Ln(GCoTim)] + \beta_{HCRR}\,[Ln(HCRR)] + \beta_{IDDCR}\,[Ln(IDDCR)]$$

The code mixed model with interaction effects (named Code Mixed –IE hereafter) is:

$$Ln(\text{Defect density in testing}) = \beta_0 + \beta_{KLOC}\,[Ln(KLOC)]$$
$$+ \beta_{DDsTim}\,[Ln(DDsTim)] + \beta_{EDRR}\,[Ln(EDRR)] + \beta_{FDDDR}\,[Ln(FDDDR)]$$
$$+ \beta_{GCoTim}\,[Ln(GCoTim)] + \beta_{HCRR}\,[Ln(HCRR)] + \beta_{IDDCR}\,[Ln(IDDCR)]$$
$$+ \beta_{KLOC*DDsTim}\,[Ln(KLOC)*Ln(DDsTim)]$$
$$+ \beta_{KLOC*EDRR}\,[Ln(KLOC)*Ln(EDRR)]$$
$$+ \beta_{KLOC*FDDDR}\,[Ln(KLOC)*Ln(FDDDR)]$$
$$+ \beta_{KLOC*IDDCR}\,[Ln(KLOC)*Ln(IDDCR)]$$
$$+ \beta_{EDRR*FDDDR}\,[Ln(EDRR)*Ln(FDDDR)]$$
$$+ \beta_{*KLOC*DDsTim*EDRR*FDDDR}[Ln(KLOC)*Ln(DDsTim)*Ln(EDRR)*Ln(FDDDR)]$$
$$+ \beta_{*KLOC*GCoTim*HCRR*IDDCR}[Ln(KLOC)*Ln(GCoTim)*Ln(HCRR)*Ln(IDDCR)]$$

Table 144 Mixed Models for Code

	Code Mixed PSPb C	Code Mixed –IE PSPb C	Code Mixed PSPb C++	Code Mixed –IE PSPb C++
Covariance Parameters	55	55	55	55
Columns in X	8	15	8	15
Columns in Z	0	0	0	0
Subjects	197	197	108	108
Observations Used	1758	1758	920	920
Prob > ChiSq	<0.0001	<0.0001	<0.0001	<0.0001
AIC	5861.4	5854.6	3084.9	3096.5

The parameter estimates of the fixed main effects for the code mixed models, and the associated standard errors, are listed in Table 145 for the data sets including influential outliers.

Table 145 Fixed-Effect Estimates for the Code Mixed Models

Parameter	Code Mixed PSPb C	Code Mixed –IE PSPb C	Code Mixed PSPb C++	Code Mixed –IE PSPb C++
Intercept	1.08**** (0.18)	0.88* (0.39)	0.60* (0.23)	0.18 (0.47)
KLnKLOC	-0.06 (0.05)	-0.17 (0.14)	-0.15* (0.07)	-0.46* (0.19)
DLnDsTim	0.17**** (0.05)	-0.09 (0.14)	0.04 (0.06)	0.30 (0.17)
ELnDRR	-0.12 (0.06)	-0.48*** (0.13)	-0.12 (0.09)	-0.16 (0.16)
FLnDDDR	0.12** (0.04)	-0.46** (0.16)	0.08 (0.05)	0.29 (0.20)
GLnCoTim	0.33**** (0.06)	0.27**** (0.06)	0.46**** (0.09)	0.37**** (0.09)
HLnCRR	-0.21*** (0.06)	-0.26*** (0.07)	-0.22* (0.09)	-0.28 (0.10)
ILnDDCR	-0.001 (0.04)	-0.60**** (0.12)	0.12 (0.06)	0.21 (0.18)

The parameter estimates of the fixed main effects for the code mixed models are consistent with those of the design mixed models, with the exception that *defect density in testing* decreases as *defect density in design review* increases for the code model with interactions, possibly as a result of interaction effects.

As *code time* increases, *defect density in testing* increases, which is consistent with the findings in Section 4.8.4. As the *code review rate* increases in terms of hours/KLOC, *defect density in testing* decreases. Given this formulation of review rate, the result is consistent with

the belief that following recommended practice will result in higher quality software. As *defect density in code review* increases, *defect density in testing* decreases, which is inconsistent with the results of Section 4.8.6, but the only statistically significant instance is for one of the models with interaction effects, and interactions may be affecting the result for the main effect.

All interactions were investigated, but only the seven interaction effects that were shown to be statistically significant for either or both of the data sets were retained. The fixed interaction effects are listed in Table 146.

Table 146 Interaction-Effect Estimates in the Code Mixed Models

Parameter	Code Mixed PSPb C	Code Mixed –IE PSPb C	Code Mixed PSPb C++	Code Mixed –IE PSPb C++
KLnKLOC *DLnDDsTim	--	-0.11* (0.05)	--	0.11 (0.06)
KLnKLOC *ELnDRR	--	-0.15*** (0.04)	--	-0.04 (0.06)
KLnKLOC *FLnDDDR	--	-0.25*** (0.07)	--	0.12 (0.09)
KLnKLOC *ILnDDCR	--	0.27**** (0.05)	--	0.11 (0.08)
ELnDRR *FLnDDDR	--	0.22** (0.07)	--	-0.04 (0.09)
KLnKLOC *DLnDsTim *ELnDRR *FLnDDDR	--	0.03*** (0.009)	--	-0.008 (0.01)
KLnKLOC *GLnCoTim *HLnCRR *ILnDDCR	--	-0.01** (0.004)	--	-0.02*** (0.005)

Program size interacts with all of the design variables individually and as a whole, as well as with all of the code variables as a whole.

The dimensions and fit statistics for the code mixed models for *defect density in testing* are shown in Table 147 for the mixed models excluding influential outliers.

Table 147 Mixed Models for Code Excluding Outliers

	Code Mixed PSPb C	Code Mixed –IE PSPb C	Code Mixed PSPb C++	Code Mixed –IE PSPb C++
Covariance Parameters	55	55	55	55
Columns in X	8	15	8	15
Columns in Z	0	0	0	0
Subjects	197	197	108	108
Observations Used	1711	1705	892	884
Prob > ChiSq	<0.0001	<0.0001	<0.0001	<0.0001
AIC	5663.5	5645.7	2968.1	2955.2

The results for the code mixed models excluding outliers are consistent with those of the code mixed models including outliers. The code mixed models excluding outliers have a consistently lower *AIC* than the models including outliers.

The parameter estimates of the fixed main effects for the code mixed models, and the associated standard errors, are listed in Table 148 for the data sets excluding outliers.

Table 148 Fixed-Effect Estimates for the Code Mixed Models Excluding Outliers

Parameter	Code Mixed PSPb C	Code Mixed –IE PSPb C	Code Mixed PSPb C++	Code Mixed –IE PSPb C++
Intercept	1.45**** (0.18)	0.92* (0.41)	0.87*** (0.23)	0.37 (0.50)
KLnKLOC	0.01 (0.05)	-0.18 (0.15)	-0.03 (0.07)	-0.36 (0.21)
DLnDsTim	0.15*** (0.05)	-0.02 (0.14)	0.05 (0.06)	0.50** (0.17)
ELnDRR	-0.09 (0.07)	-0.47*** (0.14)	-0.14 (0.09)	-0.03 (0.17)
FLnDDDR	0.11** (0.04)	-0.46** (0.16)	0.10 (0.05)	0.04 (0.21)
GLnCoTim	0.28**** (0.06)	0.26**** (0.07)	0.42**** (0.09)	0.32*** (0.09)
HLnCRR	-0.23*** (0.06)	-0.25*** (0.07)	-0.23* (0.09)	-0.29** (0.10)
ILnDDCR	-0.02 (0.04)	0.56**** (0.13)	0.11 (0.06)	0.16 (0.19)

The parameter estimates of the fixed main effects for the code mixed models excluding outliers are consistent with those of the code mixed models including outliers with the exception of *defect density in code review*, where the parameter estimate is now positive and consistent with the results of Section 4.8.6.

The fixed interaction effects are listed in Table 149.

Table 149 Interaction-Effect Estimates in the Code Mixed Models

Ln(Parameter)	Code Mixed PSPb C	Code Mixed –IE PSPb C	Code Mixed PSPb C++	Code Mixed –IE PSPb C++
KLnKLOC *DLnDsTim	--	-0.08 (0.05)	--	0.20** (0.07)
KLnKLOC *ELnDRR	--	-0.15*** (0.05)	--	0.03 (0.07)
KLnKLOC *FLnDDDR	--	-0.24*** (0.07)	--	0.009 (0.09)
KLnKLOC *ILnDDCR	--	0.25**** (0.05)	--	0.07 (0.09)
ELnDRR *FLnDDDR	--	0.21** (0.08)	--	-0.03 (0.09)
KLnKLOC *DLnDsTim *ELnDRR *FLnDDDR	--	0.03*** (0.01)	--	-0.008 (0.01)
KLnKLOC *GLnCoTim *HLnCRR *ILnDDCR	--	-0.009* (0.004)	--	-0.01** (0.005)

The parameter estimates of the fixed interaction effects for the code mixed models excluding outliers are consistent with those of the code mixed models including outliers with the exception of (*program size* * *design time*), where the sign of the interaction term has become positive.

7.4.4 Mixed Models in Compile

The dimensions and fit statistics for the compile mixed models for *defect density in testing* are shown in Table 150, where the X matrix of fixed effects contains the *program size* and the design, code, and compile process variables; the repeated measures are for the students across assignments; and there are no random effects. Influential outliers are included.

There are no random effects in this model. The repeated measures across assignments are captured in the *R* matrix for the error terms, which is contained in the SAS output in Appendix C, along with the values for the unstructured covariance matrix. The compile mixed model without interaction effects (named Compile Mixed hereafter) for the fixed effects is:

$$Ln(\text{Defect density in testing}) = \beta_0 + \beta_{KLOC}\,[Ln(KLOC)]$$
$$+ \beta_{DDsTim}\,[Ln(DDsTim)] + \beta_{EDRR}\,[Ln(EDRR)] + \beta_{FDDDR}\,[Ln(FDDDR)]$$
$$+ \beta_{GCoTim}\,[Ln(GCoTim)] + \beta_{HCRR}\,[Ln(HCRR)] + \beta_{IDDCR}\,[Ln(IDDCR)]$$
$$+ \beta_{JDDCm}\,[Ln(JDDCm)]$$

The compile mixed model with interaction effects (named Compile Mixed –IE hereafter) is:

$$Ln(\text{Defect density in testing}) = \beta_0 + \beta_{KLOC}\,[Ln(KLOC)]$$
$$+ \beta_{DDsTim}\,[Ln(DDsTim)] + \beta_{EDRR}\,[Ln(EDRR)] + \beta_{FDDDR}\,[Ln(FDDDR)]$$
$$+ \beta_{GCoTim}\,[Ln(GCoTim)] + \beta_{HCRR}\,[Ln(HCRR)] + \beta_{IDDCR}\,[Ln(IDDCR)]$$
$$+ \beta_{KLOC*DDsTim}\,[Ln(KLOC)*Ln(DDsTim)]$$
$$+ \beta_{KLOC*EDRR}\,[Ln(KLOC)*Ln(EDRR)]$$
$$+ \beta_{KLOC*FDDDR}\,[Ln(KLOC)*Ln(FDDDR)]$$
$$+ \beta_{KLOC*IDDCR}\,[Ln(KLOC)*Ln(IDDCR)]$$
$$+ \beta_{GCoTim*JDDCm}\,[Ln(GCoTim)*Ln(JDDCm)]$$
$$+ \beta_{*KLOC*GCoTim*HCRR*IDDCR}[Ln(KLOC)*Ln(GCoTim)*Ln(HCRR)*Ln(IDDCR)]$$

Table 150 Mixed Models for Compile

	Compile Mixed PSPb C	Compile Mixed –IE PSPb C	Compile Mixed PSPb C++	Compile Mixed –IE PSPb C++
Covariance Parameters	55	55	55	55
Columns in X	9	15	9	15
Columns in Z	0	0	0	0
Subjects	197	197	108	108
Observations Used	1758	1758	920	920
Prob > ChiSq	<0.0001	<0.0001	<0.0001	<0.0001
AIC	5847.3	5840.3	3077.1	3084.6

The parameter estimates of the fixed main effects for the compile mixed models, and the associated standard errors, are listed in Table 151 for the data sets including influential outliers.

Table 151 Fixed-Effect Estimates for the Compile Mixed Models

Parameter	Compile Mixed PSPb C	Compile Mixed –IE PSPb C	Compile Mixed PSPb C++	Compile Mixed –IE PSPb C++
Intercept	1.01**** (0.18)	1.44*** (0.44)	0.49* (0.23)	-0.26 (0.53)
KLnKLOC	-0.06 (0.05)	-0.22 (0.15)	-0.10 (0.07)	-0.31 (0.19)
DLnDsTim	0.16*** (0.05)	-0.03 (0.14)	0.04 (0.06)	0.28 (0.17)
ELnDRR	-0.10 (0.06)	-0.39** (0.13)	-0.09 (0.09)	-0.08 (0.16)
FLnDDDR	0.12** (0.04)	-0.18 (0.13)	0.06 (0.05)	0.16 (0.17)
GLnCoTim	0.28**** (0.06)	-0.05 (0.12)	0.41**** (0.08)	0.56*** (0.17)
HLnCRR	-0.19** (0.06)	-0.24*** (0.06)	-0.19* (0.09)	-0.25** (0.09)
ILnDDCR	-0.003 (0.04)	0.53**** (0.12)	0.11 (0.06)	0.16 (0.17)
JLnDDCm	0.11**** (0.02)	-0.12 (0.08)	0.13**** (0.03)	0.27* (0.11)

The parameter estimates of the fixed main effects for the compile mixed models are consistent with those of the design and code mixed models. *Defect density in testing* increases as *defect density in code review* increases, which is expected and consistent with the design mixed model.

As *defect density in compile* increases, *defect density in testing* increases, which is consistent with the results of Section 4.8.7. *Program size* was not shown to be statistically significant for any of the compile mixed models as a main effect but is well represented as an interaction effect.

All interactions were investigated, but only six interaction effects that were shown to be statistically significant for either or both of the data sets were retained. The interaction effects are listed in Table 152.

Table 152 Interaction-Effect Estimates in the Compile Mixed Models

Parameter	Compile Mixed PSPb C	Compile Mixed –IE PSPb C	Compile Mixed PSPb C++	Compile Mixed –IE PSPb C++
KLnKLOC *DLnDDsTim	--	-0.07 (0.05)	--	0.10 (0.06)
KLnKLOC *ELnDRR	--	-0.13** (0.04)	--	-0.01 (0.06)
KLnKLOC *FLnDDDR	--	-0.13* (0.05)	--	0.06 (0.07)
KLnKLOC *ILnDDCR	--	0.25**** (0.05)	--	0.08 (0.08)
GLnCoTim *JLnDDCm	--	0.09** (0.03)	--	-0.07 (0.04)
KLnKLOC *GLnCoTim *HLnCRR *ILnDDCR	--	-0.01** (0.004)	--	-0.02**** (0.004)

Program size interacts with all of the design variables individually, as well as with all of the compile variables as a whole.

The dimensions and fit statistics for the compile mixed models for *defect density in testing* are shown in Table 153 for the mixed models excluding influential outliers.

Table 153 Mixed Models for Compile Excluding Outliers

	Compile Mixed PSPb C	Compile Mixed –IE PSPb C	Compile Mixed PSPb C++	Compile Mixed –IE PSPb C++
Covariance Parameters	55	55	55	55
Columns in X	9	15	9	15
Columns in Z	0	0	0	0
Subjects	197	197	108	108
Observations Used	1711	1705	892	884
Prob > ChiSq	<0.0001	<0.0001	<0.0001	<0.0001
AIC	5645.2	5629.7	2960.3	2936.5

The results for the compile mixed models excluding outliers are consistent with those of the compile mixed models including outliers. The compile mixed models excluding outliers have a consistently lower *AIC* than the models including outliers.

The parameter estimates of the fixed main effects for the compile mixed models, and the associated standard errors, are listed in Table 154 for the data sets excluding outliers.

Table 154 Fixed-Effect Estimates for the Compile Mixed Models Excluding Outliers

Parameter	Compile Mixed PSPb C	Compile Mixed –IE PSPb C	Compile Mixed PSPb C++	Compile Mixed –IE PSPb C++
Intercept	1.37**** (0.18)	1.37** (0.45)	0.74** (0.23)	-0.42 (0.55)
KLnKLOC	0.03 (0.05)	-0.21 (0.15)	0.006 (0.07)	-0.18 (0.21)
DLnDsTim	0.13** (0.05)	0.04 (0.14)	0.05 (0.06)	0.48** (0.17)
ELnDRR	-0.06 (0.07)	-0.39** (0.13)	-0.12 (0.09)	0.09 (0.17)
FLnDDDR	0.11** (0.04)	-0.19 (0.13)	0.08 (0.05)	-0.07 (0.18)
GLnCoTim	0.23*** (0.06)	-0.02 (0.13)	0.36**** (0.09)	0.67**** (0.17)
HLnCRR	-0.21*** (0.06)	-0.23*** (0.07)	-0.19* (0.09)	-0.25** (0.09)
ILnDDCR	-0.02 (0.04)	0.48*** (0.13)	0.11 (0.06)	0.08 (0.18)
JLnDDCm	0.12**** (0.02)	-0.07 (0.09)	0.13**** (0.03)	0.41*** (0.12)

The parameter estimates of the fixed main effects for the compile mixed models excluding outliers are consistent with those of the compile mixed models including outliers.

The fixed interaction effects are listed in Table 155.

Table 155 Interaction-Effect Estimates in the Compile Mixed Models

Parameter	Compile Mixed PSPb C	Compile Mixed –IE PSPb C	Compile Mixed PSPb C++	Compile Mixed –IE PSPb C++
KLnKLOC *DLnDsTim	--	-0.04 (0.05)	--	0.17* (0.07)
KLnKLOC *ELnDRR	--	-0.13** (0.05)	--	0.08 (0.07)
KLnKLOC *FLnDDDR	--	-0.13* (0.05)	--	-0.04 (0.08)
KLnKLOC *ILnDDCR	--	0.22**** (0.05)	--	0.01 (0.08)
GLnCoTim *JLnDDCm	--	0.07* (0.03)	--	-0.12* (0.05)
KLnKLOC *GLnCoTim *HLnCRR *ILnDDCR	--	-0.008* (0.004)	--	-0.01* (0.005)

The parameter estimates of the fixed interaction effects for the compile mixed models excluding outliers are consistent with those of the compile mixed models including outliers.

7.4.5 Random Effects in the Mixed Models for PSP

The analyses described in Section 4.6 did not find the potentially confounding variables, such as *years of experience* or *highest degree attained*, statistically significant. In mixed models, such variables can be analyzed as random effects. A separate set of random effects models was built for each random effect investigated, using the compile mixed models to provide fixed effects. The models were run with and without interaction effects, , for data sets PSPa and PSPb, including and excluding influential outliers.

The *PSP major process* was not found to be a statistically significant random effect for any of the mixed models. This is inconsistent with the results of Section 4.5, but it is the expected result for a repeated measures mixed model across assignments, since the repeated measure incorporates a more detailed measure of change over time than the *PSP major process*.

The *number of programs finished* was not found to be a statistically significant random effect for any of the mixed models. This is consistent with the results of Section 4.6.2, which focused on the difference between those finishing all ten assignments versus those finishing less than ten.

The *PSP class* was found to be a statistically significant random effect for the mixed models that included outliers. For the data sets excluding outliers, *PSP class* was not found to be a statistically significant random effect. This is consistent with the results of Section 4.6.3: a few individuals in a small class who are struggling with an assignment can skew the performance of the class as a whole, but when those atypical cases were excluded, there were no statistically significant differences between the different offerings of the PSP course.

The *highest degree attained* was not found to be a statistically significant random effect for any of the mixed models. This is consistent with the results of Section 4.6.4.

Years of experience was not found to be a statistically significant random effect for any of the mixed models. This is consistent with the results of Section 4.6.5.

The *number of languages known* was not found to be a statistically significant random effect for any of the mixed models. This is consistent with the results of Section 4.6.6.

The *percent of time programming* in the previous year was not found to be a statistically significant random effect for any of the mixed models. This is consistent with the results of Section 4.6.7.

The *programming language* used was not found to be a statistically significant random effect for any of the mixed models. This is consistent with the results of Section 4.6.8.

Programmer ability, as measured by average *defect density in testing* for the first three assignments, was shown to be a statistically significant random effect for all of the mixed models. This is consistent with the results of Section 4.9, and is to be expected since it is a subject-specific measure, and the repeated measures across assignments in the compile mixed models capture the change in performance across PSP for the entire population. Subject-specific repeated measures are analyzed in Section 7.4.6 and provide a more sophisticated insight based on individual regression curves than the surrogate for ability used here.

7.4.6 Random Coefficient Mixed Models for Student-Specific Effects

In conventional regression theory, subject-specific terms do not occur in the model [Khattree and Naik 1999, 288-293; Littell et al. 1996, 231-232]. Estimates of the intercept and the slope are averages over the entire population, which is why the *programmer ability* variable was shown to be a statistically significant random effect.

Mixed models can support subject-specific parameters, where the model equations include random variables for the intercept and the slope. This model can be described by:

$$y_{ij} = \beta_0 + s_i + (\beta_1 + d_i) X_{ij} + e_{ij}$$

where β_0 is the fixed intercept, β_1 is the fixed slope, s_i is the random deviation of the i^{th} subject's intercept from β_0, and d_i is the random deviation of the i^{th} subject's slope from β_1. Subject-specific estimates are a form of BLUP. When the PSP *assignment* is viewed as a random coefficient, a unique regression formula can be assigned to the i^{th} student's performance for each of the ten assignments:

$$Ln[(\text{Defect density in testing})_{i,Assignment}] = \beta_0 + (\text{Student Intercept})_i + [\beta_1 + (\text{Student Slope})_i] \text{ Assignment}$$

In Figure 16, the 110 students finishing all ten assignments in (PSPb, C) are grouped into quartiles based on their individual intercepts, and the average performance of each quartile is plotted – the top quartile (*TQ*), middle top quartile (*T M2*), middle bottom quartile (*B M2*), and bottom quartile (*BQ*) provide a distinctive view on performance across the PSP class for programmers of differing abilities. This graph is similar to Figure 12, however it is based on using regression estimates to assign a quartile for each student and averaging estimated performance for each assignment within each quartile, rather than empirically assigning a *programmer quartile* based on performance on the first three assignments and averaging actual performance for each assignment within each quartile. This is a more holistic view of performance across the PSP course, but it is also estimated rather than empirical performance.

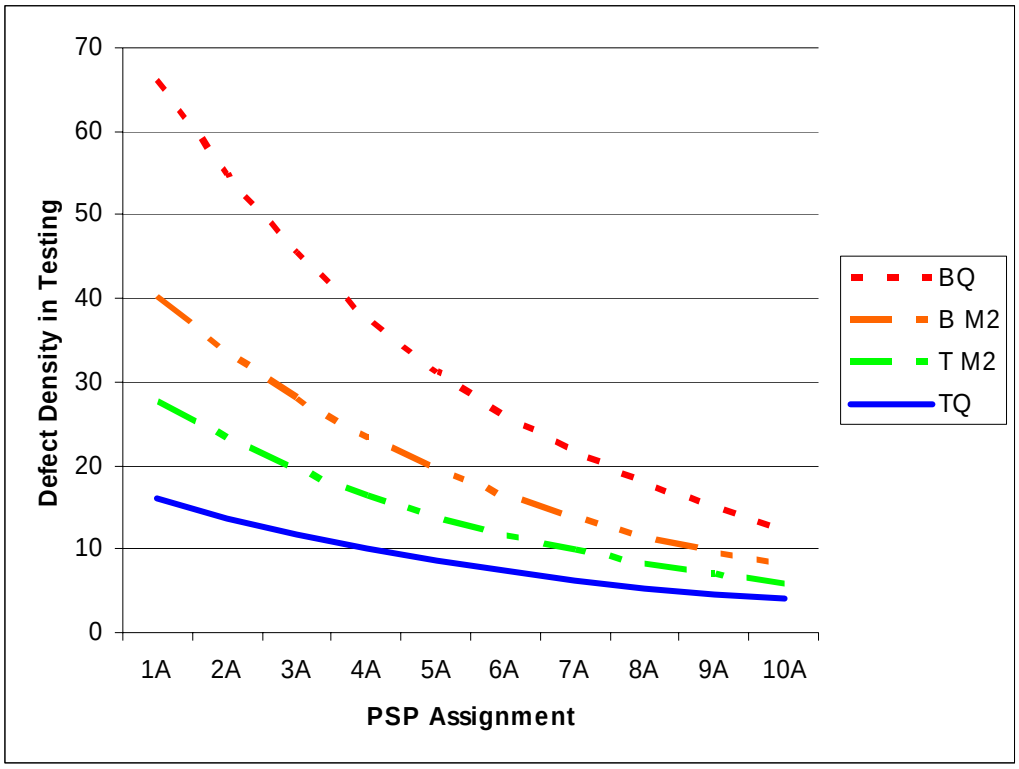

Figure 16 Trends in Software Quality from a Student-Specific Mixed Model

The slopes for all of the students are negative, ranging from -0.215 to -0.105 for (PSPb,C) and from -0.223 to -0.110 for (PSPb,C++). Note that these values are for the log transforms of *defect density in testing*. *Defect density in testing* decreased for every student finishing the course. The student-specific intercepts and slopes for the data sets (PSPb, C) and (PSPb, C++) are contained in Appendix C.

The average performance of students in the top quartile for assignments 1 and 10 is compared to that of students in the bottom quartile in Table 156.

Table 156 Comparing Top and Bottom Quartile Average Performance Based on Regression Estimates

	PSPb C TQ : BQ	PSPb C++ TQ : BQ
1A	16.4 : 66.8	14.4 : 39.8
10A	4.0 : 12.4	3.1 : 11.1
N	27	27
Percent Change	76% : 82%	78% : 72%

The top-quartile students improved 76-78%, and the bottom-quartile students improved 72-82%. Based on the regressions for the individual students, the top-quartile students improved roughly the same as the bottom-quartile students: about 4.1X to 4.6X for the top quartile and 3.6X to 5.4X for the bottom quartile. The distinction in the empirical results in Section 4.9.1 between the improvement rates for the top and bottom quartiles essentially disappeared when the random coefficients mixed model was used to generate the average performance for each quartile. This arguably makes an even stronger case for the importance of disciplined processes than the empirical comparison.

7.4.7 Discussion of the Mixed Models

The mixed models appear to be conceptually superior to the regression models. The multiplicative models perform better than the additive models, which fits the mental picture of how the process factors as described in Figure 10 earlier. The potentially confounding variables were not statistically significant as random effects, which is both desirable and consistent with the results in Chapter 4. The random coefficient model indicates that subject-specific differences are significant, which supports the intuitive understanding of "ability" as well as the exploratory data analysis and multiple regression results for the *programmer ability* surrogate.

While confirmation of the results from regression and ANOVA analyses from the more statistically sophisticated mixed models is useful, the practical value may lie in using the programmer as a random effect in building statistical models. In this context, the important result is the consistent improvement of performance across students by employing increasingly sophisticated processes. While the Hawthorne effect, that people change their behavior as a result of being observed [Parsons 1974], cannot be ruled out as a factor, systematic and sustained improvement is the management objective. Explicitly factoring in the human element into our statistical models is a possibility to be actively considered, if sufficient data can be collected, and if the potential for driving dysfunctional behavior as a result of the motivational use of measurement can be avoided.

7.5 CONCLUSIONS FOR DEFECT PREDICTION MODELS

My research demonstrates that sophisticated process-based models for defect prediction are feasible, even when using data from individual professionals where order-of-magnitude differences are to be expected. Mixed models demonstrate that individual differences are both practically and statistically significant and that factors frequently used as surrogates for ability and technology are not useful, at least for individual data. Although the amount of variation explained by the PSP models is less than might be preferred, it is still sufficient to be useful. Models using detailed process data from teams and individuals over a period of time should not suffer from the same degree of variability, although high amounts of variation are intrinsic to human-centric, design-intensive processes such as software engineering.

The unique contribution of the mixed models, in comparison to the repeated measures ANOVA performed by earlier researchers, is the ability to probe into both random effects and

the individual performance of each student. These results for the impact of PSP on software quality are consistent with those of repeated measures ANOVA in the Hayes and Over study [Hayes and Over 1997], but Hayes and Over focused on the trends at the level of the *PSP major process,* where my analyses investigated programmer ability and the underlying process factors that affected software quality.

Multiple regression models using detailed process data have the potential for being generalized, at least conceptually, for use in industry. The flexibility and power of the mixed models in dealing with random effects make them a tool worth exploring further in a team or project context.

Some researchers prefer Bayesian models that integrate expert judgment, arguing that curve fitting models are inappropriate for software because current curve fitting models are too simplistic, not addressing process-based issues. My research indicates that sophisticated process-based regression models are feasible, although they do require significant amounts of data, and there is a non-trivial cost in both money and effort in collecting the data.

A related challenge is instantiating disciplined and measured processes without driving dysfunctional behavior in those performing the work [Austin 1996]. Even in the PSP context, relatively few students arrive at a disciplined process that follows recommended practice in the class, although the measurement-based learning process continues afterwards. A motivational use of the PSP (or TSP) data, e.g. for promotions or raises, could adversely affect the validity of the data reported or the behavior of the programmers with respect to important project or organizational concerns that are not addressed by the measurement program.

My contribution in this analysis therefore consists of the following results:

- *Programmer ability*, when empirically measured, is an important factor affecting software quality that interacts with *program size* and the process variables in multiple interaction effects with two or more factors.
- Process variables, such as effort and review rate, affect software quality both as main effects and as interaction effects, allowing the development of sophisticated models when sufficient data is available, as was the case with the PSP data.
- More complete statistical models, as Fenton and Neil desired [Fenton and Neil 1999, 153], need large amounts of data and sophisticated techniques, e.g., mixed models to address random effects and individual differences effectively, as intuitively demonstrated by the contrasting results for multiplicative and additive models when building mixed models versus multiple regression models.
- When increasing process discipline is applied across the PSP assignments, individual performance with respect to software quality is consistently improved for all students as shown by the mixed models.

For managers, these results may be intuitively obvious, but my research quantifies these conclusions, at least within the context of PSP. Some programmers performed consistently better than others. Top-quartile programmers performed better than bottom-quartile programmers by a factor ranging from 2.8X to 4.1X. In all cases, disciplined processes significantly improved the quality of the work – by a conservative factor of 2X or better for top-quartile professionals; by a factor of 4X or better for bottom-quartile professionals. For individual professionals, the message is also clear: disciplined processes can improve performance.

8.0 DEFECT REMOVAL EFFECTIVENESS OF REVIEWS AND INSPECTIONS

8.1 THE RESEARCH QUESTION: DEFECT REMOVAL EFFECTIVENESS

The research in this chapter focuses on identifying factors that significantly affect the *defect removal effectiveness* of the peer review (or inspection) process. *Defect removal effectiveness* is defined as the percentage of defects eventually known to be present in a work product that are identified by a specific verification step, such as an inspection. It therefore ranges from 0% to 100%. When there are no defects in a work product at the time of the verification step, *defect removal effectiveness* is not well defined.

A number of peer review methods have been defined, from informal walkthroughs to formal inspections. Rules for effective inspections [Fagan 1976; Fagan 1986] include:

- The optimum number of inspectors is four.
- The preparation rate for inspecting code should be about 100 LOC/hour (no more than 200 LOC/hour).
- The meeting review rate for code inspections should be about 125 LOC/hour (no more than 250 LOC/hour) for code.
- Inspection meetings should not last more than two hours.

The crucial point in understanding the power of peer reviews is 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. Peer reviews can have a significant impact on the cost, quality, and development time of the software since they can be applied early in the development cycle.

In addition to the PSP data already analyzed, data from one TSP project and one high maturity project were available for this analysis, although not all of the data that might be desired was available. One of the challenges for empirical research in software engineering is obtaining valid data, and the data available for this research is limited. It is, however, sufficient for an initial analysis.

Since the focus of this analysis is on the effectiveness of the inspection process, the surrogate for software (process) quality is *defect removal effectiveness*. Analysis of *defect removal effectiveness* implies that reviews or inspections where no defects were initially present should be excluded since the effectiveness of the inspection process is a moot point in such a case.

8.2 EFFECTIVENESS OF PSP REVIEWS

The Personal Software Process (PSP) applies the concepts of process discipline and quantitative management to the work of the individual software professional in a classroom setting. PSP typically involves the development of ten programs, using increasingly sophisticated processes [Humphrey 1995]. Design and code reviews, which are personal reviews conducted by an engineer on his or her own design or code, are introduced in the 7th of the ten PSP assignments. They are designed to help engineers achieve 100% yield: all defects removed before compiling the program.

PSP reviews are a variant of inspections. Although inspections are a form of peer review, and the PSP reviews are performed by the student on his or her own code, some of the rules for effectively doing the review are likely to be applicable. The premier example is that the review

rate should be about 100 LOC/hour and no more than 200 LOC/hour. As already shown, most PSP students do not perform the reviews at this recommended rate.

This is a consequence of the PSP being a learning environment. Although students are aware of the rules for inspections, the purpose of the personal software process is to identify the most effective review process for the individual programmer. While it is likely that most students should conform to recommended practice, part of the learning experience is to experientially and quantitatively go through that learning process, therefore most students do not meet the review rate initially (and many may continue to converge to "their best personal review rate" after the PSP class is over if they continue to use the PSP's measurement-driven improvement approach for personal learning).

Both design and code reviews were analyzed. The variables of interest are *programmer ability*, as measured by average software quality on the first three PSP assignments, *program size* (in KLOC), *design time* and *coding time* (in hours/KLOC), and *review rate* (in hours/KLOC, which is the inverse of the more frequently used LOC/hour).

8.2.1 Considering Transformations for Defect Removal Effectiveness

Transformations of the *defect removal effectiveness* variable were considered, including the arcsine root and logit transformations. The arcsine root transformation is commonly used for proportions, i.e., $T_i = \arcsin(\sqrt{X_i})$ [Osborne 2002]. A logit transformation of percentage variables may also be used to approximate a normal distribution., i.e., $T_i = \log(\frac{X_i - LowerLimit}{UpperLimit - X_i})$ [Breyfogle 1999, 380].

Figure 17 illustrates the frequency distribution for *defect removal effectiveness* of PSP code reviews for (PSPb, C, NoOutliers). Data transformations can be useful in approximating a

normal distribution, which is assumed by many statistical analyses, but they work best on data skewed on one side. Data transformations alter the distance between observations in the data set, but order is maintained [Osborne 2002]. As can be observed in the figure, the histogram has peaks at 0% and 100% *defect removal effectiveness*.

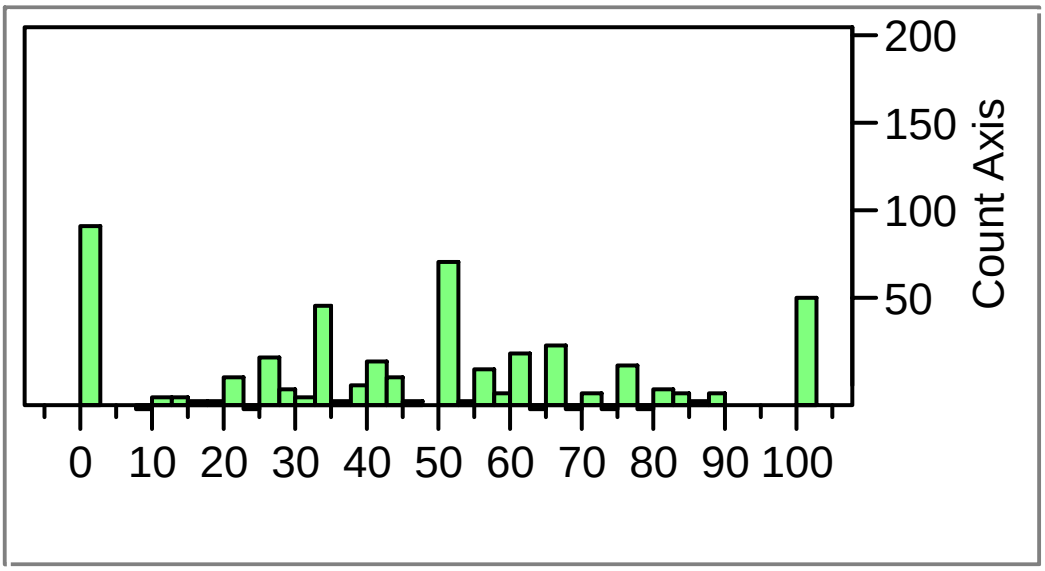

Figure 17 Distribution of *Defect Removal Effectiveness*

Figure 18 illustrates the frequency distribution for the logit transformation of defect removal effectiveness of PSP code reviews for (PSPb, C).

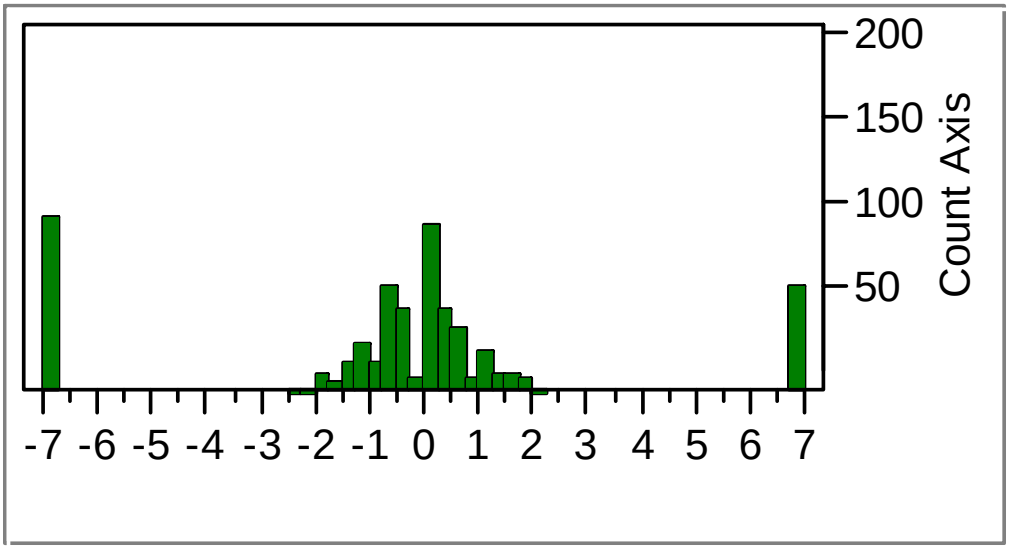

Figure 18 Distribution of Logit Transformation of *Defect Removal Effectiveness*

As can be observed for the logit transformation, the central portion of the data approximates a normal distribution more closely, but the tails at 0% and 100% prevent the transformed variable from being a good approximation of the normal distribution. The transformations did not materially improve the multiple regression models. This pattern was consistently observed for the *defect removal effectiveness* of the reviews.

The *defect removal effectiveness* variable is therefore used as the dependent variable without any transformation in the multiple regression models used to initially explore the factors affecting the reviews. Scatter diagrams and ANOVA are used to focus on specific factors.

8.2.2 Design Reviews in PSP

The multiple regression model used in the initial analysis of *defect removal effectiveness* for PSP design reviews was:

$$(\textit{Defect removal effectiveness}) = \beta_0 + \beta_{PgmrAb}(\textit{Programmer ability}) + \beta_{KLOC}(\textit{Program size}) + \beta_{DsTim}(\textit{Design time}) + \beta_{DRR}(\textit{Design review rate})$$

with the null hypotheses tested being $H_0: \beta_i=0$ with alternative hypothesis $H_a: \beta_i\neq0$. The regression results for this model are contained in Table 157. The data sets analyzed include those with and without influential outliers, as identified using leverage, studentized deleted residuals, Cook's distance, and DFFITS (see Section 7.3.7). Note that the outliers excluded in this analysis are different from those excluded in Section 4.8.2; in that section, the outliers were identified relative to *design review rate* using the interquartile limits technique.

Table 157 Multiple Regression Models for PSP Design Reviews

Source		**PSPb C including outliers**	**PSPb C excluding outliers**	**PSPb C++ including outliers**	**PSPb C++ excluding outliers**
Model	DF	4	4	4	4
	SS	1.8	2.1	1.8	2.0
	MS	0.4	0.5	0.5	0.5
Error	DF	371	356	242	233
	SS	52.7	50.8	33.0	31.2
	MS	0.1	0.1	0.1	0.1
Total	DF	375	360	246	237
	SS	54.4	52.9	34.8	33.2
F Ratio		3.1	3.6	3.3	3.8
Prob > F		0.0154	0.0065	0.0115	0.0050
R^2_a		0.0220	0.0283	0.0363	0.0454

The small values for R^2_a in these models reinforce the impact of individual differences in PSP. Larger values of R^2_a would be expected for projects if team effects decrease variability and increase predictability. This will be considered in the analysis of the TSP and high maturity projects.

The parameter estimates of the regression model for *defect removal effectiveness*, and the associated standard errors, are listed in Table 158. In the following tables, for the null hypothesis $H_0: \beta_i=0$, a *p-value*<0.05 is indicated with *, a *p-value*<0.01 is indicated with **, a *p-value*<0.001 is indicated with ***, and a *p-value*<0.0001 is indicated with ****.

Table 158 Estimates for PSP Design Review Models

Parameter	PSPb C including outliers	PSPb C excluding outliers	PSPb C++ including outliers	PSPb C++ excluding outliers
β_0 (Intercept)	0.51**** (0.05)	0.48**** (0.06)	0.59**** (0.06)	0.58**** (0.07)
Programmer Ability	-0.001* (0.0005)	-0.001* (0.0005)	-0.0005 (0.0007)	-0.0005 (0.0008)
Program Size	0.23 (0.19)	0.44 (0.25)	-0.31 (0.21)	-0.37 (0.21)
Design Time	-0.001 (0.003)	-0.001 (0.003)	-0.008* (0.003)	-0.008* (0.004)
Design Review Rate	0.02** (0.006)	0.02** (0.006)	0.02** (0.007)	0.03** (0.009)

Defect removal effectiveness consistently increases as *design review rate* increases, which is as expected since the review rate is measured in hours/KLOC. There is some evidence that *programmer ability* and *design time* also affect *defect removal effectiveness*. As *programmer ability* increases (or more accurately, as the ability of the programmer lessens), *defect removal*

effectiveness decreases, which is as expected. As *design time* increases, *defect removal effectiveness* decreases, which may indicate that difficulties in developing the design correspond to more defects in the design or a less capable programmer, as was discussed in Section 4.8.1.

It is clear from this model that *design review rate* is related to effective design reviews. This suggests that, from a statistical process control perspective, *design review rate* is a useful variable to control. This is consistent with the practices of many high maturity organizations [Florac 2000; Weller 2000].

The scatter diagram in Figure 19 illustrates the relationship between *design review rate* and *defect removal effectiveness* for (PSPb, C, NoOutliers). It is unclear from this diagram what the best rate is; the recommended *design review rate* is less than 200 LOC/hour, which is more than 5 hours/KLOC in the units used for *design review rate* in this analysis.

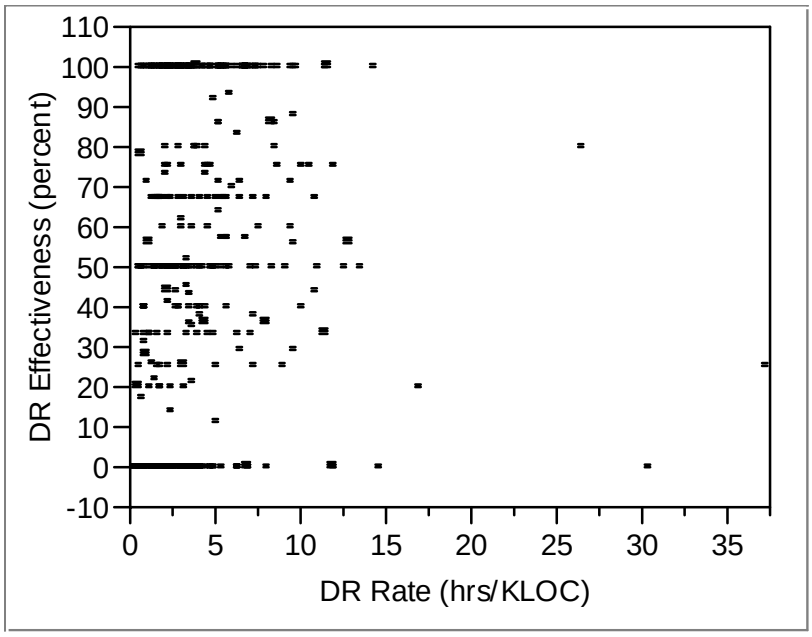

Figure 19 Scatter Diagram for *Design Review Rate* vs *Defect Removal Effectiveness*

The analysis in Section 4.8.2 is repeated with the new definition of outliers to confirm whether following recommended practice for the *design review rate* is effective. The

recommended *design review rate* is less than 200 LOC/hour. A faster rate is considered ineffective, and re-inspection should be scheduled. This provides two classes of design review based on review rate: those where the design review rate is faster than the recommended (specification) limit of 200 LOC/hour; and design reviews according to recommended practice. As illustrated in Figure 20 for (PSPb, C, NoOutliers), the classes of design review rate appear significantly different.

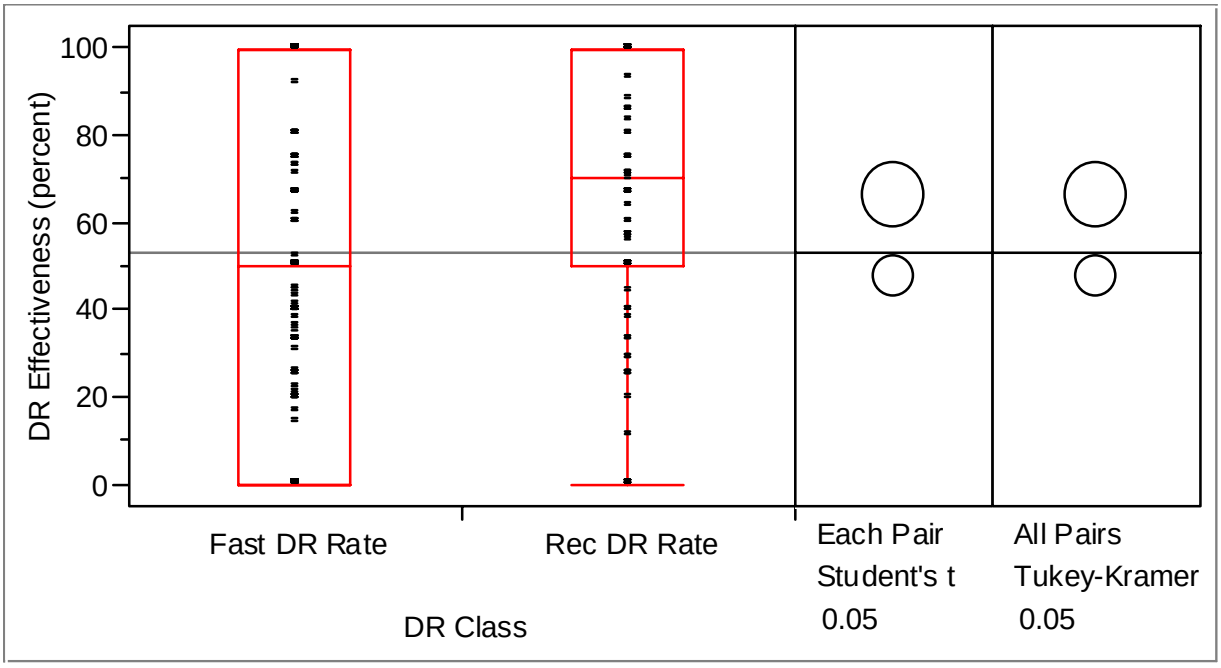

Figure 20 Differences in *Design Review Classes* Reprised

The ANOVA results for the effect of *design review class* on *defect removal effectiveness* are shown in Table 159. The null hypothesis is $H_0 : \mu_{FastDRRate} = \mu_{RecDRRate}$ with alternative hypothesis $H_a : \mu_{FastDRRate} \neq \mu_{RecDRRate}$.

Table 159 ANOVA for Regressing *Defect Removal Effectiveness* on *Design Review Class*

Source		PSPb C including outliers	PSPb C excluding outliers	PSPb C++ including outliers	PSPb C++ excluding outliers
Model	DF	1	1	1	1
	SS	23420.8	26005.4	2430.1	3722.0
	MS	23420.8	26005.4	2430.1	3722.0
Error	DF	372	357	247	238
	SS	515149.4	497166.3	350419.1	333258.3
	MS	1384.8	1392.6	1418.7	1400.2
Total	DF	373	358	248	239
	SS	538570.2	523171.7	352849.2	336980.3
F Ratio		19.2^{W}	21.8^{W}	2.2^{W}	3.6^{W}
Prob > F		$<0.0001^{W}$	$<0.0001^{W}$	0.1442^{W}	0.0609^{W}
R^2_a		0.0409	0.0470	0.0029	0.0069

The effect of *design review class* on *defect removal effectiveness* was shown to be statistically significant for two of the four data sets, although it is also close to being significant for (PSPb, C++, NoOutliers). This indicates that *design review class* may be a useful predictor variable for *defect removal effectiveness*.

The estimates of the means for *defect removal effectiveness* at the different levels of *design review class*, and the associated standard errors for the means, are listed in Table 160. The model can be expressed as:

(*Defect removal effectiveness*) = $\beta_{DR\ Class} X_{DR\ Class}$

where $\beta_{DR\ Class}$ is the level for the *design review class* and $X_{DR\ Class}$ is an indicator variable for whether that *design review class* is the correct one for the observation.

Table 160 Estimates for Regressing *Defect Removal Effectiveness* on *Design Review Class*

Levels	PSPb C including outliers**** (std err)	PSPb C excluding outliers**** (std err)	PSPb C++ including outliers (std err)	PSPb C++ excluding outliers (std err)
Fast DR Rate	48.15 (2.38)	48.15 (2.44)	49.77 (2.91)	50.46 (2.93)
Recommended DR Rate	66.57 (3.18)	67.02 (3.22)	56.92 (3.90)	59.66 (3.88)

All of the analyses, in Section 4.8.2 and here, of the effect of *design review class* on *defect removal effectiveness* provided mixed results. In the PSP context for individual programmers, it is unclear whether "following recommended practice" is "best practice." The evidence does indicate that a slower review rate leads to a more effective design review, but there is no clear-cut point for individual professionals where performance is significantly improved. This reinforces, to some degree, the PSP strategy that suggests that each student find his or her best process based on measurement and systematic improvement.

8.2.3 Code Reviews in PSP

The multiple regression model used in the initial analysis of *defect removal effectiveness* for PSP code reviews is:

$$\textit{(Defect removal effectiveness)} = \beta_0 + \beta_{PgmrAb}\,\textit{(Programmer ability)} + \beta_{KLOC}\,\textit{(Program size)} + \beta_{CoTim}\,\textit{(Code time)} + \beta_{CRR}\,\textit{(Code review rate)}$$

with the null hypotheses tested being $H_0: \beta_i=0$ with alternative hypothesis $H_a: \beta_i\neq 0$. The regression results for this model are contained in Table 161. The data sets analyzed include those with and without influential outliers. Note that the outliers excluded in this analysis are

different from those excluded in Section 4.8.5; in that section, the outliers were identified relative to *code review rate* using the interquartile limits technique.

Table 161 Multiple Regression Models for PSP Code Reviews

Source		**PSPb C including outliers**	**PSPb C excluding outliers**	**PSPb C++ including outliers**	**PSPb C++ excluding outliers**
Model	DF	4	4	4	4
	SS	5.5	5.3	2.4	3.0
	MS	1.4	1.3	0.6	0.7
Error	DF	611	593	325	315
	SS	50.4	49.8	24.0	23.0
	MS	0.08	0.08	0.07	0.07
Total	DF	615	597	329	319
	SS	55.9	55.1	26.4	26.0
F Ratio		16.8	15.7	8.1	10.2
Prob > F		<0.0001	<0.0001	<0.0001	<0.0001
R^2_a		0.0929	0.0897	0.0790	0.1033

The parameter estimates of the regression model for *defect removal effectiveness*, and the associated standard errors, are listed in Table 162.

Table 162 Estimates for PSP Code Review Models

Parameter	PSPb C including outliers	PSPb C excluding outliers	PSPb C++ including outliers	PSPb C++ excluding outliers
β_0 (Intercept)	0.42**** (0.03)	0.40**** (0.04)	0.49**** (0.04)	0.45**** (0.04)
Programmer Ability	-0.000002 (0.0003)	-0.00002 (0.0003)	-0.0007 (0.0005)	-0.0008 (0.0004)
Program Size	-0.07 (0.13)	0.03 (0.16)	-0.06 (0.14)	-0.02 (0.14)
Code Time	-0.01**** (0.002)	-0.01**** (0.002)	-0.01**** (0.003)	-0.01*** (0.003)
Code Review Rate	0.03**** (0.004)	0.04**** (0.005)	0.02**** (0.005)	0.04**** (0.006)

Defect removal effectiveness consistently increases as *code review rate* increases, which is as expected since the review rate is measured in hours/KLOC. *Defect removal effectiveness* consistently decreases as *code time* increases.

It is clear from this model that *code review rate* is related to effective code reviews. This suggests that, from a statistical process control perspective, *code review rate* is a useful variable to control. *Code time* is also a significant factor, but it is unclear what actions should be taken to control the process, since there may be design activities occurring in coding, as discussed in Sections 4.8.4 and 7.3.4.

The scatter diagram in Figure 21 illustrates the relationship between *code review rate* and *defect removal effectiveness* for (PSPb, C, NoOutliers). It is unclear from this diagram what the best rate is; the recommended *code review rate* is less than 200 LOC/hour, which is more than 5 hours/KLOC in the units used for *code review rate* in this analysis.

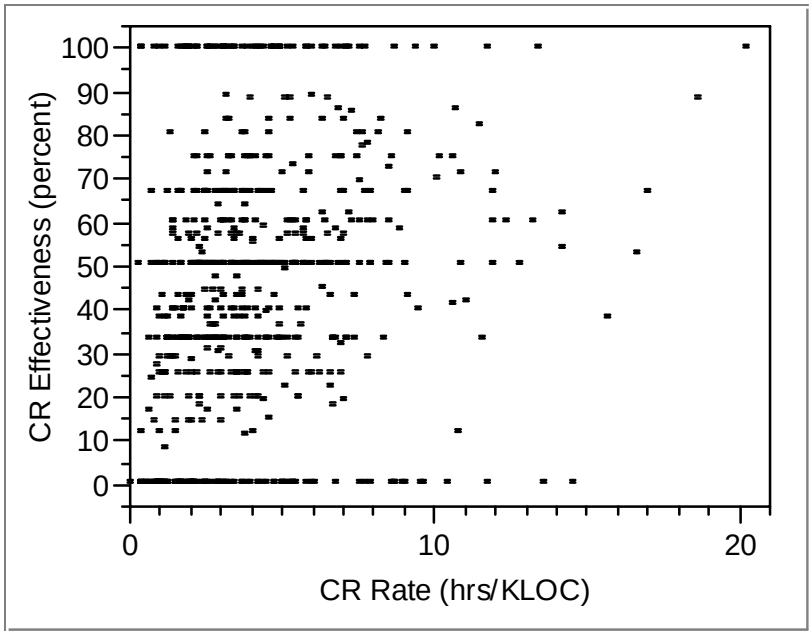

Figure 21 Scatter Diagram for *Code Review Class* vs *Defect Removal Effectiveness*

The analysis in Section 4.8.5 is repeated with the new definition of outliers to confirm whether following recommended practice for the *code review rate* is effective. The recommended *code review rate* is less than 200 LOC/hour. As illustrated in Figure 22 for (PSPb, C, NoOutliers), the classes of code review rate appear significantly different.

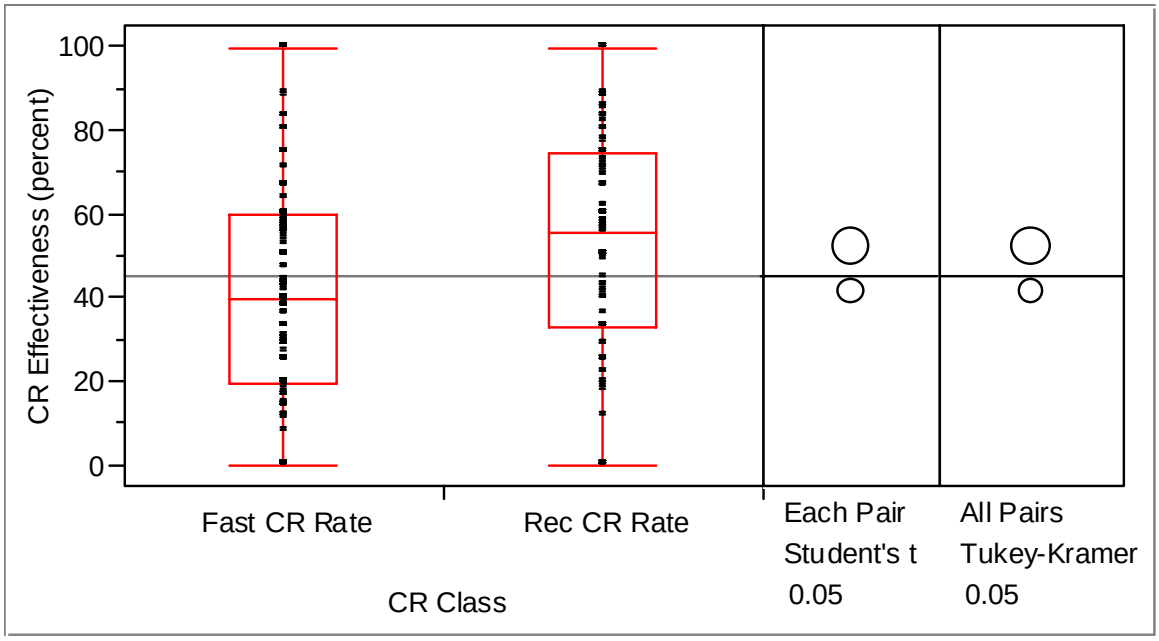

Figure 22 Differences in *Code Review Class* Reprised

The ANOVA results for the effect of *code review class* on *defect removal effectiveness* are shown in Table 163. The null hypothesis is $H_0 : \mu_{FastCRRate} = \mu_{RecCRRate}$ with alternative hypothesis $H_a : \mu_{FastCRRate} \neq \mu_{RecCRRate}$.

Table 163 ANOVA for Regressing *Defect Removal Effectiveness* on *Code Review Class*

Source		PSPb C including outliers	PSPb C excluding outliers	PSPb C++ including outliers	PSPb C++ excluding outliers
Model	DF	1	1	1	1
	SS	14509.4	13981.6	5949.0	7505.0
	MS	14509.4	13981.6	5949.0	7505.0
Error	DF	611	593	328	318
	SS	536949.6	529115.4	258606.0	252385.0
	MS	878.8	892.3	788.4	793.7
Total	DF	612	594	329	319
	SS	551459.0	543096.9	264554.9	259890.0
F Ratio		16.5	15.7	7.5	9.5
Prob > F		<0.0001	<0.0001	0.0063	0.0023
R^2_a		0.0247	0.0241	0.0195	0.0258

The effect of *code review class* on *defect removal effectiveness* was shown to be statistically significant for all of the data sets. This indicates that *code review class* is a useful predictor variable for *defect removal effectiveness*.

The estimates of the means for *defect removal effectiveness* at the different levels of *code review class*, and the associated standard errors for the means, are listed in Table 164. The model can be expressed as:

(*Defect removal effectiveness*) = $\beta_{CR\ Class} X_{CR\ Class}$

where $\beta_{CR\ Class}$ is the level for the *code review class* and $X_{CR\ Class}$ is an indicator variable for whether that *code review class* is the correct one for the observation.

Table 164 Estimates for Regressing *Defect Removal Effectiveness* on *Code Review Class*

Levels	PSPb C including outliers**** (std err)	PSPb C excluding outliers**** (std err)	PSPb C++ including outliers** (std err)	PSPb C++ excluding outliers** (std err)
Fast CR Rate	41.90 (1.44)	42.07 (1.46)	43.51 (1.84)	43.51 (1.84)
Recommended CR Rate	52.42 (2.14)	52.68 (2.25)	53.22 (2.82)	54.95 (3.02)

For code reviews, it is clearer that "following recommended practice" is desirable than it was for design reviews, even if it is unclear what the best rate is. The evidence indicates that a slower review rate leads to a more effective code review, but there is no clear-cut point for individual professionals where performance is significantly improved.

8.3 EFFECTIVENESS OF TSPS REVIEWS AND INSPECTIONS

Data from a project using the Team Software Process (TSP) was obtained. TSP is a follow-on to the PSP that incorporates the PSP concepts of measured process control and improvement and adds team-building and coordination in an industry context. TSP includes both individual reviews and inspections by the team.

The TSP project was a pilot project. The project was staffed by five programmers building 15 modules. There was a significant amount of requirements volatility (requirements

were added and deleted throughout the project) and some resource issues during the course of the pilot. As a result of the volatility in the project, there were some instances where the process was not followed, e.g., design inspections were not held on four of the 15 modules. Because there are only 15 observations for this project, only simple regression models for *defect removal effectiveness* are reported.

The variables of interest are *programmer*, encoded as A, B, C, D, and E; *program size* (in KLOC), *review rate* (in hours/KLOC for the individual), and *inspection rate* (in hours/KLOC for the team). The *inspection rates* are for meetings; the number of inspectors participating in the meeting was not reported, although it seems likely that all project members participated when available. Preparation time was not reported. Observations were excluded for modules where there were no defects present at the time of the review or inspection.

8.3.1 Impact of Program Size

The regression results for the effect of *program size* on *defect removal effectiveness* for design reviews, design inspection, code reviews, and code inspections are shown in Table 165. For the regression model:

$$(\text{Defect removal effectiveness}) = \beta_0 + \beta_1 (\text{Program size})$$

the null hypothesis tested is $H_0: \beta_1 = 0$ with alternative hypothesis $H_a: \beta_1 \neq 0$.

Table 165 Regressing *Defect Removal Effectiveness* on *Program Size* in TSP

Source		TSP Design Reviews	TSP Design Inspections	TSP Code Reviews	TSP Code Inspections
Model	DF	1	1	1	1
	SS	0.04	0.008	0.0004	0.13
	MS	0.04	0.008	0.0004	0.13
Error	DF	9	9	12	11
	SS	0.19	0.80	1.25	2.11
	MS	0.02	0.09	0.10	0.19
Total	DF	10	10	13	12
	SS	0.23	0.80	1.25	2.24
F Ratio		1.7	0.09	0.0	0.7
Prob > F		0.2314	0.7717	0.9501	0.4366
R^2_a		0.1547	0.0098	0.0003	0.0559

The effect of *program size* on *defect removal effectiveness* was consistently shown not to be statistically significant for TSP. This indicates that *program size* is not a useful predictor variable for *defect removal effectiveness*.

The parameter estimates of the regression model for *program size*, and the associated standard errors, are listed in Table 166.

Table 166 Estimates for Regressing *Defect Removal Effectiveness* on *Program Size* in TSP

Parameter	TSP Design Reviews (std err)	TSP Design Inspections (std err)	TSP Code Reviews (std err)	TSP Code Inspections (std err)
β_0 (Intercept)	0.16* (0.06)	0.77**** (0.12)	0.29* (0.11)	0.65** (0.16)
β_1	-0.17 (0.13)	-0.08 (0.27)	-0.02 (0.27)	-0.30 (0.38)

It is not surprising that *program size* is not a significant factor for *defect removal effectiveness*. One of the inspection rules for effective inspections is to limit the size of the work product being inspected so that fatigue will not affect the effectiveness of the review. Only two of the modules were large enough for this to be a concern (817 and 1154 LOC), and the related meetings did not exceed the two-hour limit.

8.3.2 Impact of the Programmer

The regression results for the effect of *programmer* on *defect removal effectiveness* for design reviews, design inspections, code reviews, and code inspections are shown in Table 167. For the *programmer*, the regression model is:

(*Defect removal effectiveness*) = $\beta_{Programmer} X_{Programmer}$

where $\beta_{Programmer}$ is the level for a specific *programmer* and $X_{Programmer}$ is an indicator variable for whether that *programmer* is the correct one for the observation. The null hypothesis against *defect removal effectiveness* is $H_0 : \mu_A = \mu_B = \mu_C = \mu_D = \mu_E$ with alternative hypothesis H_a: *not all of the means are equal*.

Table 167 Regressing *Defect Removal Effectiveness* on *Programmer* in TSP

Source		TSP Design Reviews	TSP Design Inspections	TSP Code Reviews	TSP Code Inspections
Model	DF	4	4	4	4
	SS	0.05	0.09	0.77	2.13
	MS	0.01	0.02	0.19	0.53
Error	DF	6	6	9	8
	SS	0.18	0.72	0.47	0.11
	MS	0.03	0.12	0.05	0.01
Total	DF	10	10	13	12
	SS	0.23	0.81	1.25	2.24
F Ratio		0.4	0.2	3.7	37.5
Prob > F		0.7738	0.9424	0.0486	<0.0001
R^2_a		-0.2856	-0.4911	0.4513	0.9420

The effect of *programmer* on *defect removal effectiveness* was shown to be statistically significant for code but not for design. This may indicate a systemic problem with the design process, which is perhaps also indicated by the lack of design reviews and inspections for four modules. The effect of *programmer* on *defect removal effectiveness* for code indicates that *programmer* is a useful predictor variable, but it is difficult to draw any firm conclusions given the small number of observations.

The parameter estimates of the regression model for *programmer*, and the associated standard errors, are listed in Table 168.

Table 168 Estimates for Regressing *Defect Removal Effectiveness* on *Programmer* in TSP

Parameter	TSP Design Reviews (std err)	TSP Design Inspections (std err)	TSP Code Reviews* (std err)	TSP Code Inspections**** (std err)
A	0.0 (.)	0.60 (.)	0.35 (.)	0.0 (.)
B	0.21 (0.15)	0.67 (0.33)	0.22 (0.09)	0.74 (0.12)
C	0.08 (0.06)	0.76 (0.08)	0.61 (0.19)	0.0 (0.0)
D	0.06 (0.06)	0.86 (0.05)	0.28 (0.03)	0.61 (0.11)
E	0.08 (0.08)	0.84 (0.06)	0.0 (0.0)	1.00 (0.0)

It is not surprising that *programmer* is a significant factor for *defect removal effectiveness*, but it seems more likely that there would be an observable effect for reviews than inspections since inspections are team processes and reviews are individual processes. The underlying drivers for the code review and inspection results may be not be visible in the data available, e.g., if one of the team was a superior inspector but did not participate in all inspections.

8.3.3 Impact of Review and Inspection Rates

The regression results for the effect of review / inspection rate on *defect removal effectiveness* for design reviews, design inspections, code reviews, and code inspections are shown in Table 169. For the regression model:

(Defect removal effectiveness) $= \beta_0 + \beta_1$ *(Design|code review|inspection rate)*

the null hypothesis tested is $H_0: \beta_1 = 0$ with alternative hypothesis $H_a: \beta_1 \neq 0$.

Table 169 Regressing *Defect Removal Effectiveness* on Review / Inspection Rate in TSP

Source		**TSP Design Reviews**	**TSP Design Inspections**	**TSP Code Reviews**	**TSP Code Inspections**
Model	DF	1	1	1	1
	SS	0.39	0.14	0.37	0.22
	MS	0.39	0.14	0.37	0.22
Error	DF	9	9	12	11
	SS	0.41	0.09	0.88	2.02
	MS	0.05	0.01	0.07	0.18
Total	DF	10	10	13	12
	SS	0.81	0.23	1.25	2.24
F Ratio		8.6	14.4	5.0	1.2
Prob > F		0.0169	0.0042	0.0453	0.2971
R^2_a		0.4872	0.6161	0.2937	0.0982

The effect of *review / inspection rate* on *defect removal effectiveness* was shown to be statistically significant in three of the four cases. This indicates that *review / inspection rate* is a useful predictor variable for *defect removal effectiveness*. It is also interesting to note that the R^2_a values for the statistically significant TSP models are noticeably larger than those for the PSP

models, suggesting that there is a substantive difference between the PSP and TSP processes. In part this difference may be due to the small size of the TSP data set, but the TSP models are simple regression models with only one predictor variable.

The parameter estimates of the regression model for *review / inspection rate*, and the associated standard errors, are listed in Table 170.

Table 170 Estimates for Regressing *Defect Removal Effectiveness* on Review / Inspection Rate in TSP

Parameter	TSP Design Reviews (std err)	TSP Design Inspections (std err)	TSP Code Reviews (std err)	TSP Code Inspections (std err)
β_0 (Intercept)	0.45** (0.12)	0.02 (0.04)	0.09 (0.11)	0.34 (0.24)
β_1	0.003* (0.001)	0.005** (0.001)	0.009* (0.004)	0.02 (0.02)

No design or code inspection performed by the team met the recommended rate of 200 LOC/hour or less, but almost all of the design and code reviews performed by individuals did. Comparison of conformant to nonconformant reviews or inspections is therefore not feasible. Since this was a pilot project, it seems likely that the team has the potential for continuing to learn from its data and improving performance in its inspections.

8.4 EFFECTIVENESS OF HIGH-MATURITY CODE INSPECTIONS

Data was obtained from a high maturity project that has been engaged in maintaining a high-reliability system for a number of years. The organization has been assessed at maturity

level 5 against the Software CMM. The process has changed over time, but the focus of these analyses is on code inspection data, which should be a relatively stable process.

The code inspections followed by this project are expected to follow a rigorous Fagan-style inspection [Fagan 1976; Fagan 1986]. The data reported is for modules written in a domain-specific high-order language. After removing observations with no defects present at the time of the inspection, there were 162 observations.

The variables of interest are *program size* (in KLOC), *preparation rate* (in hours/KLOC for the inspection team as a whole), *inspection rate* (in hours/KLOC), and the *number of inspectors*.

The multiple regression model used in the initial analysis of *defect removal effectiveness* for the high-maturity code reviews is:

$$\textit{(Defect removal effectiveness)} = \beta_0 + \beta_{KLOC}\,\textit{(Program size)} + \beta_{PrepRate}\,\textit{(Preparation rate)} + \beta_{CIR}\,\textit{(Code inspection rate)} + \beta_{NInsp}\,\textit{(Number of inspectors)}$$

with the null hypotheses tested being $H_0: \beta_i=0$ with alternative hypothesis $H_a: \beta_i\neq 0$. The regression results for this model are contained in Table 171. Influential outliers, as identified using leverage, studentized deleted residuals, Cook's distance, and DFFITS (see Section 7.3.7) were identified, and models were built including and excluding outliers.

Table 171 Multiple Regression Models for a High-Maturity Project

Source		High-Maturity Project including outliers	High-Maturity Project excluding outliers
Model	DF	4	4
	SS	0.42	0.93
	MS	0.10	0.23
Error	DF	157	152
	SS	27.16	25.78
	MS	0.17	0.17
Total	DF	161	156
	SS	27.58	26.71
F Ratio		0.6	1.4
Prob > F		0.6591	0.2461
R^2_a		-0.0099	0.0095

Somewhat surprisingly, neither of the models was shown to be statistically significant. This is not an expected result, especially since 31 of the 162 code inspections did not conform to the recommended code inspection rate.

This high maturity project is a maintenance project that has been involved in process improvement for a number of years in a domain where high reliability is crucial. Its quality control mechanisms are rich and diverse – problems reported after release are essentially unknown. One plausible explanation for these results is that there are so many process and quality control mechanisms in place that a minor deficiency in any one mechanism is addressed by other mechanisms, such as statistical process control. Further investigation of the *code*

inspection rate and the *number of inspectors* comprising an inspection team is appropriate, however.

The parameter estimates of the regression model for *defect removal effectiveness*, and the associated standard errors, are listed in Table 172.

Table 172 Estimates for High-Maturity Project Models

Parameter	High Maturity Project including outliers	High Maturity Project excluding outliers
β_0 (Intercept)	0.65**** (0.11)	0.67**** (0.12)
Program Size	0.03 (0.07)	-0.04 (0.14)
Preparation Rate	-0.0002 (0.0002)	0.0009 (0.0008)
Code Inspection Rate	0.0003 (0.0008)	-0.004 (0.002)
Number of Inspectors	-0.002 (0.02)	-0.001 (0.02)

None of the effects investigated were shown to be statistically significant. *Code inspection rate* and the size of the inspection team, i.e., the *number of inspectors*, are usually considered important variables, so these two factors are explored further.

8.4.1 Investigating Code Inspection Rate Further

The recommended *code inspection rate* is less than 250 LOC/hour. As illustrated in Figure 23 for the data set including influential outliers, the classes of *code inspection rate* were not shown to be statistically significant.

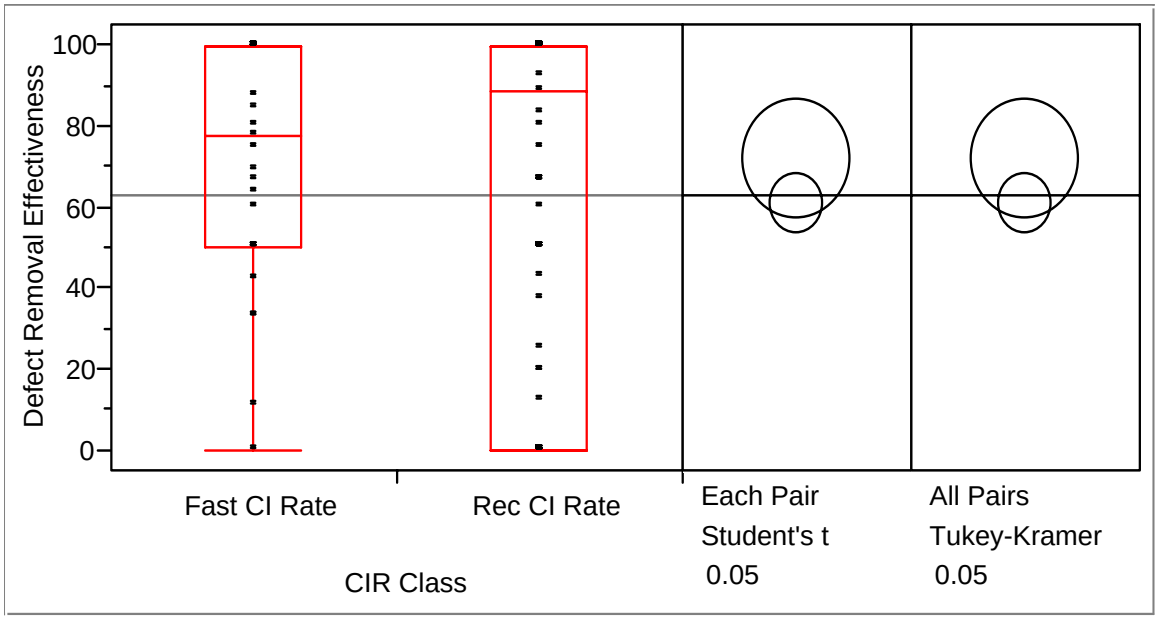

Figure 23 Differences Between *Code Inspection Classes* for a High Maturity Project

Perhaps the most interesting point about this figure is that the fast code inspections appear more effective than those following the recommended rate. The difference was not shown to be statistically significant, however. The ANOVA results for the effect of *code inspection class* on *defect removal effectiveness* are shown in Table 173. The null hypothesis is $H_0 : \mu_{FastCIR} = \mu_{RecCIR}$ with alternative hypothesis $H_a : \mu_{FastCIR} \neq \mu_{RecCIR}$.

Table 173 ANOVA for Regressing *Defect Removal Effectiveness* on *Code Inspection Class*

Source		High Maturity Project including outliers	High Maturity Project excluding outliers
Model	DF	1	1
	SS	2953.5	2817.2
	MS	2953.5	2817.2
Error	DF	160	155
	SS	272812.1	264283.7
	MS	1705.1	1705.1
Total	DF	161	156
	SS	275765.6	267100.9
F Ratio		2.8[W]	2.6[W]
Prob > F		0.0979[W]	0.1086[W]
R^2_a		0.0045	0.0042

The estimates of the means for *defect removal effectiveness* at the different levels of *code inspection class*, and the associated standard errors for the means, are listed in Table 174. The model can be expressed as:

(*Defect removal effectiveness*) = $\beta_{CI\ Class} X_{CI\ Class}$

where $\beta_{CI\ Class}$ is the level for the *code inspection class* and $X_{CI\ Class}$ is an indicator variable for whether that *code inspection class* is the correct one for the observation.

Table 174 Estimates for Regressing *Defect Removal Effectiveness* on *Code Inspection Class*

Levels	High Maturity Project including outliers (std err)	High Maturity Project excluding outliers (std err)
Fast CI Rate	72.08 (5.22)	71.57 (5.37)
Recommended CI Rate	61.23 (3.81)	60.79 (3.87)

8.4.2 Investigating Team Size Further

One of the topics of particular interest is the effect of the *number of inspectors* on *defect removal effectiveness*, but the *number of inspectors* was not shown to be statistically significant. Further exploration of this point shows that most inspection teams have between four and eight inspectors; higher values have only single data points. Only the inspections with 4-8 inspectors are considered. As illustrated in Figure 24 for the data set including influential outliers, the classes of *number of inspectors* were not shown to be statistically significant.

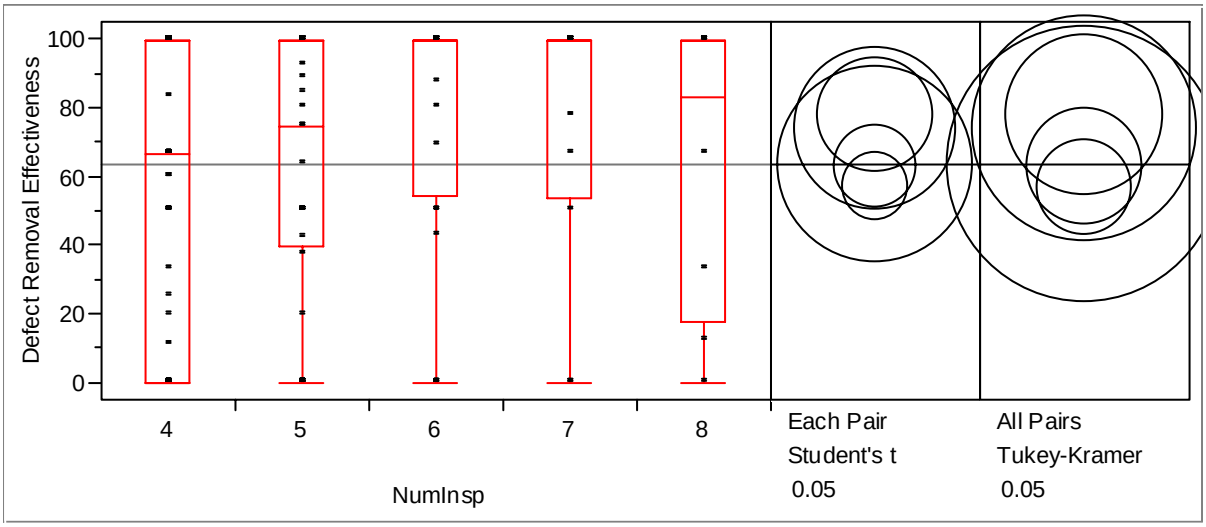

Figure 24 Differences Between *Number of Inspectors* for a High Maturity Project

Perhaps the most interesting point about this figure is that the larger teams appear to be more effective up until a size of about seven or eight team members. The difference was not shown to be statistically significant, however. The ANOVA results for the effect of *number of inspectors* on *defect removal effectiveness* are shown in Table 175. The null hypothesis is $H_0 : \mu_{TeamSize4} = \mu_{TeamSize5} = \mu_{TeamSize6} = \mu_{TeamSize7} = \mu_{TeamSize8}$ with alternative hypothesis H_a: *not all of the means are equal*.

Table 175 ANOVA for Regressing *Defect Removal Effectiveness* on *Number of Inspectors*

Source		High Maturity Project including outliers	High Maturity Project excluding outliers
Model	DF	4	4
	SS	9287.7	7785.2
	MS	2321.9	1946.3
Error	DF	153	148
	SS	256872.1	249802.6
	MS	1678.9	1687.9
Total	DF	157	152
	SS	266159.8	257587.8
F Ratio		1.5[W]	1.2
Prob > F		0.2307[W]	0.3340
R^2_a		0.0097	0.0040

The estimates of the means for *defect removal effectiveness* at the different *numbers of inspectors*, and the associated standard errors for the means, are listed in Table 176. The model can be expressed as:

(*Defect removal effectiveness*) = $\beta_{Team\ Size} X_{Team\ Size}$

where $\beta_{Team\ Size}$ is the level for the *number of inspectors* and $X_{Team\ Size}$ is an indicator variable for whether that *number of inspectors* is the correct one for the observation.

Table 176 Estimates for Regressing *Defect Removal Effectiveness* on *Number of Inspectors*

Levels	High Maturity Project including outliers (std err)	High Maturity Project excluding outliers (std err)
Number of Inspectors = 4	57.38 (5.30)	57.60 (6.36)
Number of Inspectors = 5	63.53 (5.83)	62.71 (5.90)
Number of Inspectors = 6	78.33 (7.18)	76.92 (7.77)
Number of Inspectors = 7	74.54 (11.12)	74.54 (11.12)
Number of Inspectors = 8	64.06 (15.17)	64.06 (15.17)

8.5 CONCLUSIONS FOR FACTORS AFFECTING REVIEW EFFECTIVENESS

The contribution of this analysis is to reinforce the message, which is consistent across the PSP, TSP and high maturity data, that process discipline is a major driver in software quality. Review rates are critical contributors to effective inspections. *Programmer ability* is also a crucial factor, but process performance builds beyond the foundation of competent people.

Following disciplined processes is a non-trivial achievement. As was observed in the PSP class and for the TSP project, even trained professionals can easily fall short of recommended best practices. Mature organizations that have infrastructure and culture that

support discipline provide an environment where disciplined processes can be successfully deployed – and the results demonstrate their effectiveness. As may be the case for the high maturity project investigated here, a variety of powerful quality control mechanisms reinforce and complement one another in mature processes.

This highlights the need for a continuum of approaches to process improvement. The Software Engineering Institute has addressed three tiers that need attention: the individual (PSP), the team or project (TSP), and the organization (Software CMM). Organization-focused improvement may not affect the behavior of the individual effectively. Individual-focused improvement may fail when the software professional returns to the high-pressure industry environment after participating in the PSP class.

Individual professionals need to be aware of the need for – and the benefits of – applying disciplined processes to their day-to-day work. The PSP data shows the impact of such discipline, but without reinforcement at the team and organizational level, it is easy for such discipline to slip away.

My contribution in this analysis therefore consists of the following results:

- Data transformations are of limited applicability for percentage data where there are peaks at both 0% and 100%, as is the case for defect removal effectiveness of many inspections, which implies that robust statistical techniques should be used.
- An optimal review/inspection rate is not readily apparent, although defect removal effectiveness generally improved as the rate slowed. This suggests that the recommended rate should be determined either on a per-individual basis or as part of a cost-benefit analysis that factors in issues such as the time to repair a defect at different points in the life cycle.

- An optimal team size is not readily apparent. Although defect removal effectiveness generally improved with increasing team size, the difference was not shown to be statistically significant.

For software managers, the consequences are simple to state, although they may be difficult to implement. First is the importance of competent professionals in doing high-quality work. Second, those professionals should be actively supported in following recommended practice, especially for inspections. Third, measurement-driven management is needed for the effective support of those recommended practices.

9.0 CONCLUSIONS AND RECOMMENDATIONS FOR FUTURE RESEARCH

9.1 CONCLUDING REMARKS

In recent years process discipline has moved to the forefront of software engineering concerns, largely because of concerns about software quality, and driven by the wide-spread adoption of the Software CMM. In many developing countries, credentials such as being assessed at high levels against the Software CMM are considered a requirement for being competitive [Cusumano, MacCormack, Kemerer, and Crandall 2003].

The results of my research clarify a number of outstanding empirical issues. Because the results are based on data from disciplined processes for both individuals and teams, some of the confounding factors in analyzing defect data can be separately addressed. Since the results of PSP, TSP, and CMM high maturity analyses are consistent with one another, there is reason to believe that the conclusions of my research are general with respect to the impact of using disciplined processes.

The PSP data provides a wealth of information about programmers and processes, which allows a much more comprehensive set of analyses than is typical. Although PSP data has been extensively analyzed by many different researchers, my research is unique in that it has a large enough data set to support more sophisticated models and techniques than is the norm. The richness of the data, plus the large data sets, is unusual for empirical analysis in software engineering.

In the exploratory data analysis, I demonstrated that process-based variables, *program size*, and *programmer ability* are related to software quality. Confounding variables that may appear to be plausible surrogates for areas such as ability and technology were shown not to be statistically significant within the PSP context.

I observed a notable improvement in performance and variation for software quality as students moved from *ad hoc* to disciplined processes across the PSP data sets. This supports the premise of PSP and similar process improvement strategies: disciplined software processes result in superior performance compared to *ad hoc* processes. More importantly, although empirically the top-quartile students did not improve as much as the bottom-quartile students, the 2X improvement for top-quartile performers was dramatic, demonstrating the importance of *disciplined processes* in addition to *competent professionals* as drivers of software quality.

In the outlier analysis, I demonstrated that XmR control charts using only out-of-bounds signals are roughly as effective as interquartile limits in identifying assignable causes of variation (and implying that run-based signaling techniques should be used for process control). When the stable process is identified, it may not be capable in terms of recommended practice in the PSP class, reinforcing the need for continual improvement to continue after the course. The Team Software Process (TSP) is the recommended mechanism for continuing professional development and deploying the PSP ideas in a team context.

To get the best value from statistical techniques, a consistently-implemented process is necessary but not sufficient for statistical control. The effective implementation of recommended practices is also needed before the adjective “mature” can be used. Many of the PSP students have not arrived at “industry best practice” with respect to review rates. The

objective of PSP to induce learning based on personal data, which is an on-going process that is not completed within the confines of the course.

For the statistical distributions of the PSP data, I found that the number of defects discovered in a review cannot be accurately characterized by the Poisson distribution. This is a concern since u-charts are commonly used when applying SPC to software defect data. The XmR chart, which is a robust technique in the presence of non-normal data, is a safer choice.

If techniques using distributional assumptions are used, I observed that the negative binomial distribution is one that should be considered for counts of defects. Although the results are fairly consistent for the PSP data, such assumptions should be tested against the data being analyzed in a specific case. In general, when analyzing software data, the statistical techniques used should be rigorous when distributional assumptions are violated, and whatever assumptions are made should be tested against the data.

I demonstrated that sophisticated process-based models for defect prediction are feasible even for individual software professionals. The results of the mixed models demonstrated that individual differences are both practically and statistically significant and that factors frequently used as surrogates for ability and technology are not useful, at least for individual data. The flexibility and power and power of the mixed models in dealing with random effects make them a tool worth exploring further in a team or project context.

Some analysis of project data was possible, allowing an exploration of review rates and team sizes, along with programmer differences and program size. The two factors of particular interest are the review rate and the team size since both can be controlled by the software project. Review rates that follow recommended best practice usually have superior performance, although whether the boundary for establishing the recommendation is optimal is not clear from

these empirical results. Similarly, although defect removal effectiveness increases as inspection team size grows from four to six inspectors, the difference is not statistically significant. For PSP, TSP, and the high maturity contexts, while classical statistical analyses provide useful insights, the variability is sufficiently high, even in the presence of disciplined processes, that the concerns of those who prefer non-parametric approaches, Bayesian Belief Networks, and a "relaxed" use of statistics are understandable.

My contributions in this research are summarized in the following results:

- Disciplined processes were shown to improve individual performance in software quality by a factor of about five, similar to the results of previous researchers analyzing PSP data [Hayes and Over 1997, 22; Wesslen 2000; Wohlin 2004].
- Individual differences of more than an order-of-magnitude were shown to remain even when disciplined processes were used [Ferguson et al. 1997; Hayes and Over 1997, 22; Hayes 1998; Hirmanpour and Schofield 2003; Holmes 2003; Wohlin 2004, 212].
- *Programmer ability* was shown to significantly affect software quality when empirically measured; surrogates such as *years of experience* were not found to be useful, although some earlier researchers have found team-based experience significant [Takahashi and Kamayachi 1985; Zhang 1999].
- Top-quartile performers were shown to improve by a factor of two or more; bottom-quartile performers were shown to improve by a factor of four or more.
- Although technology factors, i.e., *programming language used*, may affect productivity, they were not shown to affect software quality as measured by *defect*

density in testing, unlike some earlier researchers in a project/team environment [Gaffney 1984; Lipow 1982].

- *Program size* was shown to be a weak predictor of quality in the presence of individual differences, unlike the findings of most previous researchers in a project/team environment [Akiyama 1972; Compton and Withrow 1990; Criscione, Ferree, and Porter 2001; Halstead 1977, 87-91; Jones 1996; Lipow 1982; Lyu 1996; Fenton and Neil 1999; Fenton and Ohlsson 2000; Putnam and Myers 1997, 32].
- Detailed process measures provide more insight into performance than broad categories such as *PSP major process* or CMM maturity level, partially addressing Fenton and Neil's desire for more complete models [Fenton and Neil 1999, 153], at least within the context of individual programmers.
- PSP processes are not statistically capable or stable by the end of the ten assignments in the PSP class.
- For a retrospective analysis, XmR control charts using only out-of-bounds signals are roughly equivalent to interquartile limits in identifying outliers. This suggests that run-based signaling techniques should also be used to identify assignable causes of variation.
- When the natural process limits for a PSP process are identified, the measured process performance may not meet recommended practice, reinforcing the need for continual improvement to continue after the course. The Team Software Process is the recommended mechanism for continuing professional development and deploying the PSP ideas in a team context [Humphrey 1999].

- Statistical assumptions, such as the distribution that data follow, should be tested where they are important for achieving correct conclusions, i.e., the statistical techniques making the assumption cannot be characterized as robust when the assumptions are violated.
- The frequently made assumption that defect data follow a Poisson distribution is not valid for the PSP defect data; a negative binomial distribution is preferable.
- Although u-charts may be commonly used in the software industry for defect data [Paulk, Goldenson, and White 2000, 58-59], their use is questionable unless the statistical assumption of a Poisson distribution has been tested.
- *Programmer ability*, when empirically measured, is an important factor affecting software quality that interacts with *program size* and the process variables in multiple interaction effects with two or more factors.
- Process variables, such as effort and review rate, affect software quality both as main effects and as interaction effects, allowing the development of sophisticated models when sufficient data is available, as was the case with the PSP data.
- More complete statistical models, as Fenton and Neil desired [Fenton and Neil 1999, 153], need large amounts of data and sophisticated techniques, e.g., mixed models to address random effects and individual differences effectively, as intuitively demonstrated by the contrasting results for multiplicative and additive models when building mixed models versus multiple regression models.
- When increasing process discipline is applied across the PSP assignments, individual performance with respect to software quality is consistently improved for all students as shown by the mixed models.

- Data transformations are of limited applicability for percentage data where there are peaks at both 0% and 100%, as is the case for defect removal effectiveness of many inspections, which implies that robust statistical techniques should be used.
- An optimal review/inspection rate is not readily apparent, although defect removal effectiveness generally improved as the rate slowed. This suggests that the recommended rate should be determined either on a per-individual basis or as part of a cost-benefit analysis that factors in issues such as the time to repair a defect at different points in the life cycle.
- An optimal team size is not readily apparent. Although defect removal effectiveness generally improved with increasing team size, the difference was not shown to be statistically significant.

In conclusion, the major contribution of my research is that process discipline is a major driver in software quality that managers can proactively address. Programmer ability is also a crucial factor, but process performance builds beyond the foundation of competent people. Following disciplined processes is a non-trivial achievement. Even trained professionals can easily fall short of recommended best practices.

Mature organizations that have infrastructure and culture that support discipline provide an environment where disciplined processes can be successfully deployed – and the results demonstrate their effectiveness. My research repeatedly identifies the need for software managers and professionals who are interested in building high-quality products to focus on:

- hiring competent professionals, because there are order-of-magnitude differences in performance between people;

- supporting those professionals in consistently implementing good software engineering practices such as inspections because it is difficult to follow recommended practice;
- using measurement to control the disciplined process, because the measurement feedback is necessary for implementing recommend practice;
- ensuring that all workers exercise discipline, because even top performers improve their performance significantly by consistently implementing recommended practices (with the caveat that better practices may be identified for both the software engineering discipline and the individual as time goes by);
- using statistics to control and improve the process after the proper foundation of discipline and measurement has been put in place, because the statistical techniques will primarily identify conformance problems without the foundation.

The actions necessary to build high-quality software products may be simple, but they are not easy to implement. Quantifying the impact, as my research does, supports those managers and professionals who wish to follow disciplined processes but must address the challenges and resistance endemic to the demanding field we work in.

9.2 LIMITATIONS

The bulk of my research used classroom data. Although empirical research in software engineering frequently uses classroom data, the challenges for generalizing the results to industry projects are notable. TSP and high maturity projects offer opportunities for exploring software engineering with relatively reliable data, but gaining access to useful and reliable industry data is difficult. The project data analyzed here can be considered indicative but not definitive.

9.3 FUTURE RESEARCH DIRECTIONS

Replicating these analyses with industry data is desirable. TSP projects in particular have the potential to provide data of similar richness as the PSP data. Inspections provide, in principle, an even richer set of data than the reviews in PSP, although as has been observed here, obtaining data from conformant inspections may be a greater challenge than obtaining industry data.

Further exploration of the interactions between programmer ability and process discipline in the arena of agile methods [Boehm and Turner 2004] would illuminate a number of controversial issues, as well as reinforcing the applicability of discipline in the agile approach. Objectively defining "ability" would be a prerequisite to such a series of studies. Given the emphasis of the agile methods on people over process, an empirical investigation of the issues raised in my results for agile methods would provide a useful complement to the PSP analyses.

Similar analyses to these for software quality could be performed for productivity. The PSP data highlights an issue with how productivity is frequently defined: measures such as LOC/hour are clearly inadequate measures of productivity given the program size ranges for each PSP assignment. Proposed alternative size measures, such as function points, also appear inadequate. This definitional issue would need to be adequately addressed as a prerequisite to productivity analyses on the PSP data.

The distributional analysis could be explored more comprehensively. While the primary point – that the Poisson distribution of software defect data is questionable, and use of the u-chart may be not justified – was made, other distributions could be considered that might be superior to the negative binomial. The possibility could be explored that software defect data is

zero-inflated, i.e., there is a large number of zeroes in the data that are generated by a different process than the positive counts [Khoshgoftaar and Szabo 2001].

The use of control charts remains a controversial topic in software engineering, but the use of statistical process control is growing, as encouraged by the Software CMM and similar process improvement frameworks. A resolution of what signaling rules are appropriate for software processes depends on causal analysis, which was not feasible in the retrospective studies of the PSP data. Initiating statistical process control with the basic "point outside the control limits" is a reasonable start, but, as my research suggests, further exploration of which signal rules add value – and of which control chart techniques are appropriate – would benefit the software community.

APPENDIX A

Descriptions of Variables in Data Sets

A.1 VARIABLES IN THE PSP DATA

Note that SAS variable names for process variables have a unique first letter to prevent confusion when multiple variables interact in an effect.

Assignment	PSP Assignment (1, 2, 3, … 10)
AvgDDT	Average Defect Density in Testing for 1A-3A
ClassID	PSP Class
DDsTim	Design Time (hrs/KLOC)
Degree	Highest Degree Attained
DfInCm	Defects Injected in Compile
DfInCo	Defects Injected in Coding
DfInCR	Defects Injected in CR
DfInDR	Defects Injected in DR
DfInDs	Defects Injected in Design
DfInPl	Defects Injected in Planning
DfInTs	Defects Injected in Testing
DfRmCm	Defects Removed in Compile
DfRmCo	Defects Removed in Coding
DfRmCR	Defects Removed in Code Review

DfRmDR	Defects Removed in Design Review
DfRmDs	Defects Removed in Design
DfRmPl	Defects Removed in Planning
DfRmTs	Defects Removed in Test
DLnDsTim	Log Transform of Design Time
dreCR	Defect Removal Effectiveness of Code Review (0-1)
dreDR	Defect Removal Effectiveness of Design Review (0-1)
EDRR	Design Review Rate (hrs/KLOC)
ELnDRR	Log Transform of Design Review Rate
FDDDR	Defect Density in Design Review (defects/KLOC)
FLnDDDR	Log Transform of Defect Density in Design Review
GCoTim	Coding Time (hrs/KLOC)
GLnCoTim	Log Transform of Coding Time
HCRR	Code Review Rate (hrs/KLOC)
HLnCRR	Log Transform of Code Review Rate
IDDCR	Defect Density in Code Review (defects/KLOC)
ILnDDCR	Log Transform of Defect Density in Code Review
JDDCm	Defect Density in Compile (defects/KLOC)
JLnDDCm	Log Transform of Defect Density in Compile
KLnKLOC	Log Transform of KLOC
KLOC	Program Size (Thousands of Lines of Code)
Lang	Programming Language (C, C++, Java, VisualBasic)
LOC	Program Size (Lines of Code)
MajPrcs	PSP Major Process (PSP0, PSP1, PSP2, PSP3)
NDfCR	Number of Defects at Code Review

NDfDR	Number of Defects at Design Review
NLang	Number of Languages Known
PercSWTi	Percent of Software Time in Previous Year (0-1)
PgmCnt	Count of Assignments Finished (1, 2, 3, … 10)
PmgrAb	Programmer Ability (Average Defect Density in Testing for 1A-3A)
Program	PSP Assignment (1A, 2A, 3A, … 10A)
PSPa	PSPa Data Set Indicator (y/n)
Quartiles	Programmer Quartiles (TQ, M2, BQ)
Student	Student
StuNum	Student Number
TDDTs	Defect Density in Testing (defects/KLOC)
TimCm	Time Compile (min)
TimCo	Time Coding (min)
TimCR	Time CR (min)
TimDR	Time DR (min)
TimDs	Time Design (min)
TimPl	Time Planning (min)
TimPM	Time Postmortem (min)
TimTs	Time Testing (min)
TLnDDTs	Log Transform of Defect Density in Testing
YrsExp	Years of Experience

A.2 VARIABLES IN THE TSP PROJECT DATA

CIR	Code Inspection Rate (team) (hrs/KLOC)
CRR	Code Review Rate (individual) (hrs/KLOC)
ddCI	Code Inspection Defect Density (defects/KLOC)
ddCR	Code Review Defect Density (defects/KLOC)
ddDLDI	Design Inspection Defect Density (defects/KLOC)
ddDLDR	Design Review Defect Density (defects/KLOC)
DLDIR	Design Inspection Rate (team) (hrs/KLOC)
DLDRR	Design Review Rate (individual) (hrs/KLOC)
dreCI	Code Inspection Defect Removal Effectiveness (0-1)
dreCR	Code Review Defect Removal Effectiveness (0-1)
dreDLDI	Design Inspection Defect Removal Effectiveness (0-1)
dreDLDR	Design Review Defect Removal Effectiveness (0-1)
KLOC	Program Size (Thousands of Lines of Code)
Pgmr	Programmer

A.3 VARIABLES IN THE HIGH-MATURITY PROJECT DATA

CIR	Code Inspection Rate (hrs/KLOC)
ddCI	Defect Density in Code Inspections

	(defects/KLOC)
dreCI	Defect Removal Effectiveness of Code Inspections (0-1)
InspDefects	Number of Defects Found in Inspection
InspNum	Inspection ID Number
KLOC	Program Size (Thousands of Lines of Code)
LOC	Lines of Code
MtgTime	Meeting Time (hrs)
NDefects	Number of Defects Present
NumReqdIn	Number of Required Inspectors
PrepRate	Preparation Rate (hrs/KLOC)
PrepTime	Preparation Time (hrs)

APPENDIX B

SAS Code

B.1 GENERAL LINEAR MODELS FOR PSP

```
TITLE1 'Mark Paulk -- PSP Data Analysis';

DATA pspData;
INFILE 'c:\SASData\RawPSP.txt' LRECL=512;

INPUT
ClassID $ StuNum Student $ YrsExp PercSWTi Degree $ NLang Lang $
Assignment Program $ LOC
TimPl TimDs TimDR TimCo TimCR TimCm TimTs TimPM
DfInPl DfInDs DfInDR DfInCo DfInCR DfInCm DfInTs
DfRmPl DfRmDs DfRmDR DfRmCo DfRmCR DfRmCm DfRmTs
PgmCnt QPgmr Quartiles $ PSP1997 $;

/* Set the right data split - PSP1997 vs PSP2001 and C vs C++ usually. */

TITLE2 'GLM REGRESSION MODELS -- PSP Data Analysis for PSPb C';
IF PSP1997 ^= 'n' THEN DELETE;
IF Lang ^= 'C' THEN DELETE;

LABEL
ClassID = 'PSP Class'
StuNum = 'Student Number'
Student = 'Student'
YrsExp = 'Years of Experience'
PercSWTi = 'Percent of Software Time in Previous Year'
Degree = 'Highest Degree Achieved'
NLang = 'Number of Languages Known'
Lang = 'Programming Language'
Assignment = 'PSP Assignment'
Program = 'PSP Assignment'

TimPl = 'Time Planning (min)'
TimDs = 'Time Design (min)'
TimDR = 'Time DR (min)'
TimCo = 'Time Coding (min)'
TimCR = 'Time CR (min)'
TimCm = 'Time Compile (min)'
TimTs = 'Time Testing (min)'
TimPM = 'Time Postmortem (min)'

DfInPl = 'Defects Injected in Planning'
DfInDs = 'Defects Injected in Design'
DfInDR = 'Defects Injected in DR'
DfInCo = 'Defects Injected in Coding'
```

```
DfInCR = 'Defects Injected in CR'
DfInCm = 'Defects Injected in Compile'
DfInTs = 'Defects Injected in Testing'

DfRmPl = 'Defects Removed in Planning'
DfRmDs = 'Defects Removed in Design'
DfRmDR = 'Defects Removed in Design Reviews'
DfRmCo = 'Defects Removed in Coding'
DfRmCR = 'Defects Removed in Code Reviews'
DfRmCm = 'Defects Removed in Compile'
DfRmTs = 'Defects Removed in Test'

PgmCnt = 'Count of Assignments Finished'
QPgmr = 'Average DD in Testing 1A-3A'
Quartiles = 'Programmer Ability Quartiles'
PSP1997 = 'PSP1997 Data Set';

/* Missing values for YearsExp, PercSWTi, and NLang are encoded as -1.
   Missing (and too rare) values for Degree and Language are encoded as "Unknown".
   In some instances, students reported spending zero time for an activity such as design; the missing values for
Prodvty, DsPerc, and CoPerc are encoded as -1.
   Note that for review effectiveness, -1 is a code for "no review" and -2 is a code for "no defects to be found." */

IF YrsExp = -1 THEN YrsExp = .;
IF PercSWTi = -1 THEN PercSWTi = .;
IF NLang = -1 THEN NLang = .;
IF Degree = 'Unknown' THEN Degree =.;
IF Lang = 'Unknown' THEN Lang =.;

/* Calculate assignment variables. */

KLOC = LOC / 1000;

IF Assignment = 1 THEN MajPrcs = 0;
ELSE IF Assignment = 2 THEN MajPrcs = 0;
ELSE IF Assignment = 3 THEN MajPrcs = 0;
ELSE IF Assignment = 4 THEN MajPrcs = 1;
ELSE IF Assignment = 5 THEN MajPrcs = 1;
ELSE IF Assignment = 6 THEN MajPrcs = 1;
ELSE IF Assignment = 7 THEN MajPrcs = 2;
ELSE IF Assignment = 8 THEN MajPrcs = 2;
ELSE IF Assignment = 9 THEN MajPrcs = 2;
ELSE MajPrcs = 3;

/* Calculate design variables. */

DDsTim = (TimDs / 60) / KLOC;
EDRR = (TimDR / 60) / KLOC;
FDDDR = DfRmDR / KLOC;

LABEL
DDsTim = 'Design Time (hrs) / KLOC'
EDRR = 'DR Rate (hrs/KLOC)'
FDDDR = 'DD in DR (defects/KLOC)';

/* Calculate coding variables. */
```

```
GCoTim = (TimCo / 60) / KLOC;
HCCR = (TimCR / 60) / KLOC;
IDDCR = DfRmCR / KLOC;

LABEL
GCoTim = 'Coding Time (hrs) / KLOC'
HCCR = 'CR Rate (hrs/KLOC)'
IDDCR = 'DD in CR (defects/KLOC)';

/* Calculate compile variables. */

JDDCm = DfRmCm / KLOC;

LABEL
JDDCm = 'DD in Compile (defects/KLOC)';

/* Calculate testing variables. */

TDDTs = DfRmTs / KLOC;

LABEL
TDDTs = 'DD in Testing (defects/KLOC)';
RUN;

/* GLM regression models with interactions. */

TITLE3 'GLM design models using STEPWISE variables';
PROC GLM DATA= pspData;
MODEL TDDTs = QPgmr KLOC DDsTim EDRR FDDDR
/solution;
RUN;

PROC GLM DATA= pspData;
MODEL TDDTs = QPgmr KLOC DDsTim EDRR FDDDR
QPgmr*KLOC QPgmr*DDsTim QPgmr*EDRR QPgmr*FDDDR
DDsTim*FDDDR EDRR*FDDDR
QPgmr*KLOC*EDRR QPgmr*DDsTim*EDRR QPgmr*EDRR*FDDDR
QPgmr*KLOC*FDDDR KLOC*DDsTim*EDRR DDsTim*EDRR*FDDDR
QPgmr*DDsTim*EDRR*FDDDR QPgmr*KLOC*DDsTim*EDRR*FDDDR
/solution;
RUN;

TITLE3 'GLM code models using STEPWISE variables';
PROC GLM DATA= pspData;
MODEL TDDTs = QPgmr KLOC DDsTim EDRR FDDDR GCoTim HCCR IDDCR
/solution;
RUN;

PROC GLM DATA= pspData;
MODEL TDDTs = QPgmr KLOC DDsTim EDRR FDDDR GCoTim HCCR IDDCR
QPgmr*KLOC QPgmr*EDRR QPgmr*FDDDR QPgmr*GCoTim
QPgmr*HCCR QPgmr*IDDCR KLOC*HCCR
EDRR*GCoTim EDRR*HCCR HCCR*IDDCR
QPgmr*KLOC*GCoTim QPgmr*KLOC*HCCR QPgmr*DDsTim*EDRR
QPgmr*EDRR*HCCR QPgmr*FDDDR*IDDCR
```

```
QPgmr*KLOC*DDsTim*EDRR*FDDDR QPgmr*KLOC*GCoTim*HCCR*IDDCR
/solution;
RUN;

TITLE3 'GLM compile models using STEPWISE variables';
PROC GLM DATA= pspData;
MODEL TDDTs = QPgmr KLOC DDsTim EDRR FDDDR GCoTim HCCR IDDCR JDDCm
/solution;
RUN;

PROC GLM DATA= pspData;
MODEL TDDTs = QPgmr KLOC DDsTim EDRR FDDDR GCoTim HCCR IDDCR JDDCm
QPgmr*KLOC QPgmr*GCoTim QPgmr*HCCR QPgmr*IDDCR QPgmr*JDDCm
KLOC*GCoTim KLOC*HCCR KLOC*JDDCm GCoTim*JDDCm HCCR*IDDCR
QPgmr*KLOC*GCoTim QPgmr*KLOC*IDDCR QPgmr*KLOC*JDDCm
QPgmr*FDDDR*IDDCR FDDDR*IDDCR*JDDCm
QPgmr*KLOC*GCoTim*JDDCm QPgmr*FDDDR*IDDCR*JDDCm
/solution;
RUN;
```

B.2 INFLUENTIAL OUTLIERS FOR PSP

```
/* Calculating leverage and other influential outlier stats. */

TITLE3 'GLM compile models using STEPWISE variables';
PROC GLM DATA= pspData;
MODEL TDDTs = PgmrAb KLOC DDsTim EDRR FDDDR GCoTim HCCR IDDCR JDDCm;
OUTPUT out=OutCm p=yhat h=lever cookd=cook dffits=dff press=prs rstudent=rstd;
RUN;

PROC GLM DATA= pspData;
MODEL TDDTs = PgmrAb KLOC DDsTim EDRR FDDDR GCoTim HCCR IDDCR JDDCm
PgmrAb*KLOC PgmrAb*GCoTim PgmrAb*HCCR PgmrAb*IDDCR PgmrAb*JDDCm
KLOC*GCoTim KLOC*HCCR KLOC*JDDCm GCoTim*JDDCm HCCR*IDDCR
PgmrAb*KLOC*GCoTim PgmrAb*KLOC*IDDCR PgmrAb*KLOC*JDDCm
PgmrAb*FDDDR*IDDCR FDDDR*IDDCR*JDDCm
PgmrAb*KLOC*GCoTim*JDDCm PgmrAb*FDDDR*IDDCR*JDDCm;
OUTPUT out=OutCmIE p=yhat h=lever cookd=cook dffits=dff press=prs rstudent=rstd;
RUN;

/* Identify influential outliers */

TITLE3 'Identify influential outliers for compile';
DATA IDCm;
SET OutCm;
IF Lang = 'C' AND lever < 0.5 AND rstd < 3.291 AND dff < 0.151 AND cook < 1.03 THEN DELETE;
IF Lang = 'C++' AND lever < 0.5 AND rstd < 3.291 AND dff < 0.209 AND cook < 1.03 THEN DELETE;
RUN;
PROC PRINT DATA=IDCm;
VAR Student Program yhat lever cook dff prs rstd;
RUN;
```

```
TITLE3 'Identify influential outliers for compile IE';
DATA IDCmIE;
SET OutCmIE;
IF Lang = 'C' AND lever < 0.5 AND rstd < 3.291 AND dff < 0.248 AND cook < 1.03 THEN DELETE;
IF Lang = 'C++' AND lever < 0.5 AND rstd < 3.291 AND dff < 0.343 AND cook < 1.03 THEN DELETE;
RUN;
PROC PRINT DATA=IDCmIE;
VAR Student Program yhat lever cook dff prs rstd;
RUN;

/* Remove influential outliers */

TITLE3 'GLM compile models without influential outliers';

DATA XCm;
SET OutCm;
IF lever > 0.5 THEN DELETE;
IF rstd > 3.291 THEN DELETE;
IF cook > 1.03 THEN DELETE;
IF Lang = 'C' AND (dff > 0.151) THEN DELETE;
IF Lang = 'C++' AND (dff > 0.209) THEN DELETE;
RUN;

PROC GLM DATA= XCm;
MODEL TDDTs = PgmrAb KLOC DDsTim EDRR FDDDR GCoTim HCCR IDDCR JDDCm
/solution;
RUN;

TITLE3 'GLM compile models IE without influential outliers';
DATA XCmIE;
SET OutCmIE;
IF lever > 0.5 THEN DELETE;
IF rstd > 3.291 THEN DELETE;
IF cook > 1.03 THEN DELETE;
IF Lang = 'C' AND (dff > 0.248) THEN DELETE;
IF Lang = 'C++' AND (dff > 0.343) THEN DELETE;
RUN;

PROC GLM DATA= XCmIE;
MODEL TDDTs = PgmrAb KLOC DDsTim EDRR FDDDR GCoTim HCCR IDDCR JDDCm
PgmrAb*KLOC PgmrAb*GCoTim PgmrAb*HCCR PgmrAb*IDDCR PgmrAb*JDDCm
KLOC*GCoTim KLOC*HCCR KLOC*JDDCm GCoTim*JDDCm HCCR*IDDCR
PgmrAb*KLOC*GCoTim PgmrAb*KLOC*IDDCR PgmrAb*KLOC*JDDCm
PgmrAb*FDDDR*IDDCR FDDDR*IDDCR*JDDCm
PgmrAb*KLOC*GCoTim*JDDCm PgmrAb*FDDDR*IDDCR*JDDCm
/solution;
RUN;
```

B.3 MIXED MODELS (DESIGN, CODE, COMPILE) FOR PSP

```
TITLE3 'Multiplicative mixed models for design including outliers';
PROC MIXED data=pspData mmeq;
```

```
CLASS Student Assignment;
MODEL TLnDDTs = KLnKLOC DLnDsTim ELnDRR FLnDDDR
/ ddfm=satterth solution;
REPEATED Assignment / type=un sub=Student r rcorr;
RUN;

TITLE3 'Multiplicative mixed models for design excluding outliers';
PROC MIXED data=XCm mmeq;
CLASS Student Assignment;
MODEL TLnDDTs = KLnKLOC DLnDsTim ELnDRR FLnDDDR
/ ddfm=satterth solution;
REPEATED Assignment / type=un sub=Student r rcorr;
RUN;

TITLE3 ' Multiplicative mixed models for code including outliers';
PROC MIXED data=pspData;
CLASS Student Assignment;
MODEL TLnDDTs = KLnKLOC DLnDsTim ELnDRR FLnDDDR GLnCoTim HLnCRR ILnDDCR
/ ddfm=satterth solution;
REPEATED Assignment / type=un sub=Student r rcorr;
RUN;

TITLE3 ' Multiplicative mixed models for code with interactions including outliers';
PROC MIXED data=pspData;
CLASS Student Assignment;
MODEL TLnDDTs = KLnKLOC DLnDsTim ELnDRR FLnDDDR GLnCoTim HLnCRR ILnDDCR
KLnKLOC*DLnDsTim KLnKLOC*ELnDRR KLnKLOC*FLnDDDR KLnKLOC*ILnDDCR
ELnDRR*FLnDDDR
KLnKLOC*DLnDsTim*ELnDRR*FLnDDDR KLnKLOC*GLnCoTim*HLnCRR*ILnDDCR
/ ddfm=satterth solution;
REPEATED Assignment / type=un sub=Student r rcorr;
RUN;

TITLE3 ' Multiplicative mixed models for code excluding outliers';
PROC MIXED data=XCm;
CLASS Student Assignment;
MODEL TLnDDTs = KLnKLOC DLnDsTim ELnDRR FLnDDDR GLnCoTim HLnCRR ILnDDCR
/ ddfm=satterth solution;
REPEATED Assignment / type=un sub=Student r rcorr;
RUN;

TITLE3 ' Multiplicative mixed models for code with interactions excluding outliers';
PROC MIXED data=XCmIE;
CLASS Student Assignment;
MODEL TLnDDTs = KLnKLOC DLnDsTim ELnDRR FLnDDDR GLnCoTim HLnCRR ILnDDCR
KLnKLOC*DLnDsTim KLnKLOC*ELnDRR KLnKLOC*FLnDDDR KLnKLOC*ILnDDCR
ELnDRR*FLnDDDR
KLnKLOC*DLnDsTim*ELnDRR*FLnDDDR KLnKLOC*GLnCoTim*HLnCRR*ILnDDCR
/ ddfm=satterth solution;
REPEATED Assignment / type=un sub=Student r rcorr;
RUN;

TITLE3 'Multiplicative mixed models for compile including outliers';
PROC MIXED data= pspData;
CLASS Student Assignment;
MODEL TLnDDTs = KLnKLOC DLnDsTim ELnDRR FLnDDDR GLnCoTim HLnCRR ILnDDCR JLnDDCm
```

```
/ ddfm=satterth solution;
REPEATED Assignment / type=un sub=Student r rcorr;
RUN;

TITLE3 'Multiplicative mixed models for compile with interactions including outliers';
PROC MIXED data= pspData;
CLASS Student Assignment;
MODEL TLnDDTs = KLnKLOC DLnDsTim ELnDRR FLnDDDR GLnCoTim HLnCRR ILnDDCR JLnDDCm
KLnKLOC*DLnDsTim KLnKLOC*ELnDRR KLnKLOC*FLnDDDR KLnKLOC*ILnDDCR
GLnCoTim*JLnDDCm
KLnKLOC*GLnCoTim*HLnCRR*ILnDDCR
/ ddfm=satterth solution;
REPEATED Assignment / type=un sub=Student r rcorr;
RUN;

TITLE3 'Multiplicative mixed models for compile excluding outliers';
PROC MIXED data= XCm;
CLASS Student Assignment;
MODEL TLnDDTs = KLnKLOC DLnDsTim ELnDRR FLnDDDR GLnCoTim HLnCRR ILnDDCR JLnDDCm
/ ddfm=satterth solution;
REPEATED Assignment / type=un sub=Student r rcorr;
RUN;

TITLE3 'Multiplicative mixed models for compile with interactions excluding outliers';
PROC MIXED data= XCmIE;
CLASS Student Assignment;
MODEL TLnDDTs = KLnKLOC DLnDsTim ELnDRR FLnDDDR GLnCoTim HLnCRR ILnDDCR JLnDDCm
KLnKLOC*DLnDsTim KLnKLOC*ELnDRR KLnKLOC*FLnDDDR KLnKLOC*ILnDDCR
GLnCoTim*JLnDDCm
KLnKLOC*GLnCoTim*HLnCRR*ILnDDCR
/ ddfm=satterth solution;
REPEATED Assignment / type=un sub=Student r rcorr;
RUN;
```

B.4 MIXED MODELS WITH A RANDOM EFFECT FOR PSP

```
/* Multiplicative mixed models */

TITLE3 'Testing random variable with compile mixed model including outliers';
PROC MIXED data= pspData;
CLASS Student Assignment;
MODEL TLnDDTs = KLnKLOC DLnDsTim ELnDRR FLnDDDR GLnCoTim HLnCRR ILnDDCR JLnDDCm
/ ddfm=satterth solution;
REPEATED Assignment / type=un sub=Student r rcorr;
RANDOM PgmrAb /solution;
RUN;

TITLE3 'Testing random variable with compile mixed model -IE including outliers';
PROC MIXED data= pspData;
CLASS Student Assignment;
MODEL TLnDDTs = KLnKLOC DLnDsTim ELnDRR FLnDDDR GLnCoTim HLnCRR ILnDDCR JLnDDCm
KLnKLOC*DLnDsTim KLnKLOC*ELnDRR KLnKLOC*FLnDDDR KLnKLOC*ILnDDCR
```

```
GLnCoTim*JLnDDCm KLnKLOC*GLnCoTim*HLnCRR*ILnDDCR
/ ddfm=satterth solution;
REPEATED Assignment / type=un sub=Student r rcorr;
RANDOM PgmrAb /solution;
RUN;

TITLE3 'Testing random variable with compile mixed model excluding outliers';
PROC MIXED data= XCm;
CLASS Student Assignment;
MODEL TLnDDTs = KLnKLOC DLnDsTim ELnDRR FLnDDDR GLnCoTim HLnCRR ILnDDCR JLnDDCm
/ ddfm=satterth solution;
REPEATED Assignment / type=un sub=Student r rcorr;
RANDOM PgmrAb /solution;
RUN;

TITLE3 'Testing random variable with compile mixed model -IE excluding outliers';
PROC MIXED data= XCmIE;
CLASS Student Assignment;
MODEL TLnDDTs = KLnKLOC DLnDsTim ELnDRR FLnDDDR GLnCoTim HLnCRR ILnDDCR JLnDDCm
KLnKLOC*DLnDsTim KLnKLOC*ELnDRR KLnKLOC*FLnDDDR KLnKLOC*ILnDDCR
GLnCoTim*JLnDDCm KLnKLOC*GLnCoTim*HLnCRR*ILnDDCR
/ ddfm=satterth solution;
REPEATED Assignment / type=un sub=Student r rcorr;
RANDOM PgmrAb /solution;
RUN;
```

B.5 DEFECT REMOVAL EFFECTIVENESS FOR PSP

```
DATA DRData;
SET pspData;
IF NDfDR = 0 THEN DELETE;  /* Delete any reviews where there were no defects */
IF TimDR = 0 THEN DELETE;  /* Delete assignments where there were no reviews */
RUN;

DATA DRNoOut;
SET XCm;  /* Start without outliers */
IF NDfDR = 0 THEN DELETE;  /* Delete any reviews where there were no defects */
IF TimDR = 0 THEN DELETE;  /* Delete assignments where there were no reviews */
RUN;

TITLE3 'Defect Removal Effectiveness - Design Regression Models';
TITLE4 'Design DRE including outliers';
PROC GLM DATA= DRData;
MODEL dreDR = PgmrAb KLOC DDsTim EDRR
/solution;
OUTPUT out=ErrDs residual=err;
RUN;

TITLE4 'Design DRE without outliers';
PROC GLM DATA= DRNoOut;
MODEL dreDR = PgmrAb KLOC DDsTim EDRR
/solution;
```

```
OUTPUT out=ErrDsNo residual=err;
RUN;

DATA CRData;
SET pspData;
IF NDfCR = 0 THEN DELETE;  /* Delete any reviews where there were no defects */
IF TimCR = 0 THEN DELETE;  /* Delete assignments where there were no reviews */
RUN;

DATA CRNoOut;
SET XCm; /* Start without outliers */
IF NDfCR = 0 THEN DELETE;  /* Delete any reviews where there were no defects */
IF TimCR = 0 THEN DELETE;  /* Delete assignments where there were no reviews */
RUN;

TITLE3 'Defect Removal Effectiveness - Code Regression Models';
TITLE4 'Code DRE including outliers';
PROC GLM DATA= CRData;
MODEL dreCR = PgmrAb KLOC GCoTim HCRR
/solution;
OUTPUT out=ErrCod residual=err;
RUN;

TITLE4 'Code DRE without outliers';
PROC GLM DATA= CRNoOut;
MODEL dreCR = PgmrAb KLOC GCoTim HCRR
/solution;
OUTPUT out=ErrCodNo residual=err;
RUN;
```

B.6 TSP PROJECT MODELS

```
TITLE1 'Mark Paulk -- TSP Data Analysis';

DATA tspData;
INFILE 'c:\SASData\TSP1.txt' LRECL=512;

INPUT
Module $ Pgmr $
DfInRq DfInRI DfInHLD DfInHLDI DfInDLD DfInDLDR
DfInDLDI DfInCode DfInCR DfInCm DfInCI DfInUT DfInBnI DfInST
DfRmRq DfRmRI DfRmHLD DfRmHLDI DfRmDLD DfRmDLDR
DfRmDLDI DfRmCode DfRmCR DfRmCM DfRmCI DfRmUT DfRmBnI DfRmST
DLDLines LOC
TimDLD TimDLDR TimDLDI TimCode TimCR TimCm TimCI TimUT;

Injected = DfInRq + DfInRI + DfInHLD + DfInHLDI +
DfInDLD + DfInDLDR + DfInDLDI +
DfInCode + DfInCR + DfInCm + DfInCI +
DfInUT + DfInBnI + DfInST;

Removed = DfRmRq + DfRmRI + DfRmHLD + DfRmHLDI +
```

```
DfRmDLD + DfRmDLDR + DfRmDLDI +
DfRmCode + DfRmCR + DfRmCM + DfRmCI +
DfRmUT + DfRmBnI + DfRmST;

IF LOC = 0 THEN DELETE;
KLOC = LOC / 1000;

/* Calculate design variables. */

DLDRR = TimDLDR / KLOC;  /* Detailed design reviews (individual) */
ddDLDR = DfRmDLDR / KLOC;
NDfDLDR = DfInRq + DfInRI + DfInHLD + DfInHLDI + DfInDLD + DfInDLDR -
        DfRmRq - DfRmRI - DfRmHLD - DfRmHLDI - DfRmDLD;
IF NDfDLDR > 0 THEN dreDLDR = DfRmDLDR / NDfDLDR; ELSE dreDLDR = .;

LABEL
DLDRR = 'Design Review Rate (individual) (hrs/KLOC)'
ddDLDR = 'DLDR Defect Density (defects/KLOC)'
dreDLDR = 'DLDR Defect Removal Effectiveness (0-1)';

DLDIR = TimDLDI / KLOC;  /* Detailed design inspections (team) */
AvgDLDIR = DLDIR / 5;
ddDLDI = DfRmDLDI / KLOC;
NDfDLDI = NDfDLDR + DfInDLDI - DfRmDLDR;
IF NDfDLDI > 0 THEN dreDLDI = DfRmDLDI / NDfDLDI; ELSE dreDLDI = .;

LABEL
DLDIR = 'Design Inspection Rate (team) (hrs/KLOC)'
ddDLDI = 'DI Defect Density (defects/KLOC)'
dreDLDI = 'DI Defect Removal Effectiveness (0-1)';

/* Calculate coding and compile variables. */

CRR = TimCR / KLOC;  /* Code reviews (individual) */
ddCR = DfRmCR / KLOC;
NDfCR = NDfDLDI + DfInCode + DfInCR - DfRmDLDI - DfRmCode;
IF NDfCR > 0 THEN dreCR = DfRmCR / NDfCR; ELSE dreCR = .;

LABEL
CRR = 'Code Review Rate (individual) (hrs/KLOC)'
ddCR = 'CR Defect Density (defects/KLOC)'
dreCR = 'CR Defect Removal Effectiveness (0-1)';

ddCm = DfRmCm / KLOC; /* Compile */
NDfCm = NDfCR + DfInCm - DfRmCR;
IF NDfCm > 0 THEN dreCm = DfRmCm / NDfCm; ELSE dreCm = .;

CIR = TimCI / KLOC;  /* Code inspections (team) */
AvgCIR = CIR / 5;
ddCI = DfRmCI / KLOC;
NDfCI = NDfCm + DfInCI - DfRmCm ;
IF NDfCI > 0 THEN dreCI = DfRmCI / NDfCI; ELSE dreCI = .;

LABEL
CIR = 'Code Inspection Rate (team) (hrs/KLOC)'
ddCI = 'CI Defect Density (defects/KLOC)'
```

```
dreCI = 'CI Defect Removal Effectiveness (0-1)';

/* Calculate testing variables. */

ddTs = (DfRmUT + DfRmBnI + DfRmST) / KLOC;
NDfTs = NDfCI + DfInUT + DfInBnI + DfInST - DfRmCI ;

RUN;

TITLE3 'TSP1 Design Reviews (individual)';
PROC GLM DATA= tspData;
MODEL dreDLDR = KLOC /solution;
RUN;

PROC GLM DATA= tspData;
CLASS Pgmr;
MODEL dreDLDR = Pgmr /solution;
RUN;

PROC GLM DATA= tspData;
MODEL dreDLDR = DLDRR /solution;
RUN;

TITLE3 'TSP1 Design Inspections (team)';
PROC GLM DATA= tspData;
MODEL dreDLDI = KLOC /solution;
RUN;

PROC GLM DATA= tspData;
CLASS Pgmr;
MODEL dreDLDI = Pgmr /solution;
RUN;

PROC GLM DATA= tspData;
MODEL dreDLDI = DLDIR /solution;
RUN;

TITLE3 'TSP1 Code Reviews (individual)';
PROC GLM DATA= tspData;
MODEL dreCR = KLOC /solution;
RUN;

PROC GLM DATA= tspData;
CLASS Pgmr;
MODEL dreCR = Pgmr /solution;
RUN;

PROC GLM DATA= tspData;
MODEL dreCR = CRR /solution;
RUN;

TITLE3 'TSP1 Code Inspections (team)';
PROC GLM DATA= tspData;
MODEL dreCI = KLOC /solution;
RUN;
```

```
PROC GLM DATA= tspData;
CLASS Pgmr;
MODEL dreCI = Pgmr/solution;
RUN;

PROC GLM DATA= tspData;
MODEL dreCI = CIR/solution;
RUN;
```

B.7 HIGH-MATURITY PROJECT MODELS

```
TITLE1 'Mark Paulk -- High Maturity Project Data Analysis';

DATA HMData;
INFILE 'c:\SASData\HM1.txt' LRECL=512;

INPUT
InspNum MtgTime NumCh PrepTime LOC NDefects InspDefects NumInsp NumOptIn;

LABEL
InspNum = 'Inspection ID Number'
MtgTime = 'Meeting Time (hrs)'
NumInsp = 'Number of Checklists'
PrepTime = 'Preparation Time (hrs)'
LOC = 'Lines of Code'
NDefects = 'Number of Defects Present'
InspDefects = 'Number of Defects Found in Inspection'
NumInsp = 'Number of Inspectors'
NumOptIn = 'Number of Optional Inspectors';

IF LOC = 0 THEN DELETE;
KLOC = LOC / 1000;
IF NDefects = 0 THEN DELETE;
IF PrepTime = 0 THEN DELETE;
IF MtgTime = 0 THEN DELETE;

PrepRate = (PrepTime / NumInsp) / KLOC;
StdPrep = LOC / (PrepTime / NumInsp);
CIR = MtgTime / KLOC;
StdCIR = LOC / MtgTime;
ddCI = InspDefects / KLOC;
dreCI = InspDefects / NDefects;

LABEL
PrepRate = 'Preparation Rate (hrs/KLOC)'
StdPrep = 'Preparation Rate (LOC/hr)'
CIR = 'Code Inspection Rate (hrs/KLOC)'
StdCIR = 'Code Inspection Rate (LOC/hr)'
ddCI = 'Defect Density in CI (defects/KLOC)'
dreCI = 'Defect Removal Effectiveness (0-1)';
RUN;
```

```
TITLE3 'HM1 Code Inspections';
PROC GLM DATA= HMData;
MODEL dreCI = KLOC PrepRate CIR NumInsp /solution;
OUTPUT out=RegStats p=yhat h=lever cookd=cook dffits=dff press=prs rstudent=rstd;
RUN;

DATA Conform;
SET HMData;
IF StdPrep > 200 THEN DELETE;
IF StdCIR > 250 THEN DELETE;
RUN;

TITLE3 'HM1 Conformant Code Inspections';
PROC GLM DATA= Conform;
MODEL dreCI = KLOC PrepRate CIR NumInsp /solution;
RUN;

TITLE3 'Identify Influential Outliers';
DATA IDOut;
SET RegStats;
IF lever < 0.5 AND rstd < 3.291 AND dff < 0.3514 AND cook < 1.15 THEN DELETE;
RUN;
PROC PRINT DATA=IDOut;
VAR InspNum dreCI KLOC PrepRate CIR NumInsp;
RUN;
DATA NoOut;
SET RegStats;
IF lever > 0.5 THEN DELETE;
IF rstd > 3.291 THEN DELETE;
IF cook > 1.15 THEN DELETE;
IF dff > 0.3514 THEN DELETE;
RUN;

TITLE3 'HM1 Code Inspections without Influential Outliers';
PROC GLM DATA= NoOut;
MODEL dreCI = KLOC PrepRate CIR NumInsp /solution;
RUN;

TITLE3 'HM1 Code Inspections with Interactions';
PROC GLM DATA= HMData;
MODEL dreCI = KLOC PrepRate CIR NumInsp /solution;
OUTPUT out=RegStats p=yhat h=lever cookd=cook dffits=dff press=prs rstudent=rstd;
RUN;

TITLE3 'HM1 Code Inspections with Interactions without Influential Outliers';
PROC GLM DATA= NoOut;
MODEL dreCI = KLOC PrepRate CIR NumInsp /solution;
RUN;
```

APPENDIX C

SAS Output

C.1 COMPILE REGRESSION MODEL FOR (PSPB, C)

The GLM Procedure

Number of observations 1758

Dependent Variable: TDDTs DD in Testing (defects/KLOC)

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	9	1002755.877	111417.320	113.96	<.0001
Error	1748	1708982.871	977.679		
Corrected Total	1757	2711738.748			

R-Square	Coeff Var	Root MSE	TDDTs Mean
0.369783	98.96412	31.26786	31.59515

Source	DF	Type I SS	Mean Square	F Value	Pr > F
QPgmr	1	542630.3977	542630.3977	555.02	<.0001
KLOC	1	108539.3555	108539.3555	111.02	<.0001
DDsTim	1	1031.1429	1031.1429	1.05	0.3046
EDRR	1	82529.5492	82529.5492	84.41	<.0001
FDDDR	1	548.0905	548.0905	0.56	0.4541
GCoTim	1	96301.1321	96301.1321	98.50	<.0001
HCRR	1	52765.3203	52765.3203	53.97	<.0001
IDDCR	1	9158.7602	9158.7602	9.37	0.0022
JDDCm	1	109252.1285	109252.1285	111.75	<.0001

Source	DF	Type III SS	Mean Square	F Value	Pr > F
QPgmr	1	371531.4448	371531.4448	380.01	<.0001
KLOC	1	29284.4003	29284.4003	29.95	<.0001
DDsTim	1	1308.1471	1308.1471	1.34	0.2475
EDRR	1	108.9308	108.9308	0.11	0.7386
FDDDR	1	2175.6647	2175.6647	2.23	0.1359
GCoTim	1	32080.7130	32080.7130	32.81	<.0001
HCRR	1	10649.6419	10649.6419	10.89	0.0010
IDDCR	1	7126.4152	7126.4152	7.29	0.0070
JDDCm	1	109252.1285	109252.1285	111.75	<.0001

Parameter	Estimate	Standard Error	t Value	Pr > |t|
Intercept	9.01033915	2.01697875	4.47	<.0001
QPgmr	0.37233356	0.01909996	19.49	<.0001
KLOC	-56.27032737	10.28156949	-5.47	<.0001
DDsTim	0.10205399	0.08822666	1.16	0.2475
EDRR	-0.13582316	0.40690838	-0.33	0.7386
FDDDR	0.12238609	0.08204162	1.49	0.1359
GCoTim	0.37236571	0.06500486	5.73	<.0001
HCRR	-1.54679942	0.46866729	-3.30	0.0010
IDDCR	-0.14386592	0.05328691	-2.70	0.0070
JDDCm	0.16029801	0.01516391	10.57	<.0001

C.2 COMPILE REGRESSION MODEL WITH INTERACTIONS FOR (PSPB, C)

The GLM Procedure

Number of observations 1758

Dependent Variable: TDDTs DD in Testing (defects/KLOC)

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	26	1270016.183	48846.776	58.65	<.0001
Error	1731	1441722.565	832.884		
Corrected Total	1757	2711738.748			

R-Square	Coeff Var	Root MSE	TDDTs Mean
0.468340	91.34230	28.85973	31.59515

Source	DF	Type I SS	Mean Square	F Value	Pr > F
QPgmr	1	542630.3977	542630.3977	651.51	<.0001
KLOC	1	108539.3555	108539.3555	130.32	<.0001
DDsTim	1	1031.1429	1031.1429	1.24	0.2660
EDRR	1	82529.5492	82529.5492	99.09	<.0001
FDDDR	1	548.0905	548.0905	0.66	0.4174
GCoTim	1	96301.1321	96301.1321	115.62	<.0001
HCRR	1	52765.3203	52765.3203	63.35	<.0001
IDDCR	1	9158.7602	9158.7602	11.00	0.0009
JDDCm	1	109252.1285	109252.1285	131.17	<.0001
QPgmr*KLOC	1	101843.6726	101843.6726	122.28	<.0001
QPgmr*GCoTim	1	29479.7567	29479.7567	35.39	<.0001
QPgmr*HCRR	1	33262.0662	33262.0662	39.94	<.0001
QPgmr*IDDCR	1	6453.6212	6453.6212	7.75	0.0054
QPgmr*JDDCm	1	4805.6870	4805.6870	5.77	0.0164
KLOC*GCoTim	1	0.1321	0.1321	0.00	0.9900
KLOC*HCRR	1	1294.9884	1294.9884	1.55	0.2126

KLOC*JDDCm	1	3020.8334	3020.8334	3.63	0.0570
GCoTim*JDDCm	1	13232.1198	13232.1198	15.89	<.0001
HCRR*IDDCR	1	24272.5344	24272.5344	29.14	<.0001
QPgmr*KLOC*GCoTim	1	20927.8369	20927.8369	25.13	<.0001
QPgmr*KLOC*IDDCR	1	18531.3765	18531.3765	22.25	<.0001
QPgmr*KLOC*JDDCm	1	662.0601	662.0601	0.79	0.3727
QPgmr*FDDDR*IDDCR	1	5254.1937	5254.1937	6.31	0.0121
FDDDR*IDDCR*JDDCm	1	1669.7826	1669.7826	2.00	0.1570
QPgm*KLOC*JCoT*JDDCm	1	967.5751	967.5751	1.16	0.2813
QPgm*FDDD*MDDC*JDDCm	1	1582.0693	1582.0693	1.90	0.1683

Dependent Variable: TDDTs DD in Testing (defects/KLOC)

Source	DF	Type III SS	Mean Square	F Value	Pr > F
QPgmr	1	78200.77163	78200.77163	93.89	<.0001
KLOC	1	2.48667	2.48667	0.00	0.9564
DDsTim	1	8.38981	8.38981	0.01	0.9201
EDRR	1	273.59240	273.59240	0.33	0.5666
FDDDR	1	75.25084	75.25084	0.09	0.7638
GCoTim	1	1052.37319	1052.37319	1.26	0.2611
HCRR	1	2633.65478	2633.65478	3.16	0.0755
IDDCR	1	3089.47517	3089.47517	3.71	0.0543
JDDCm	1	7195.91308	7195.91308	8.64	0.0033
QPgmr*KLOC	1	4503.17033	4503.17033	5.41	0.0202
QPgmr*GCoTim	1	34385.20851	34385.20851	41.28	<.0001
QPgmr*HCRR	1	9019.30676	9019.30676	10.83	0.0010
QPgmr*IDDCR	1	16245.52724	16245.52724	19.51	<.0001
QPgmr*JDDCm	1	1706.91635	1706.91635	2.05	0.1524
KLOC*GCoTim	1	1864.82457	1864.82457	2.24	0.1348
KLOC*HCRR	1	15.79952	15.79952	0.02	0.8905
KLOC*JDDCm	1	1005.14511	1005.14511	1.21	0.2721
GCoTim*JDDCm	1	14715.96285	14715.96285	17.67	<.0001
HCRR*IDDCR	1	23823.34169	23823.34169	28.60	<.0001
QPgmr*KLOC*GCoTim	1	12207.59706	12207.59706	14.66	0.0001
QPgmr*KLOC*IDDCR	1	10601.52352	10601.52352	12.73	0.0004
QPgmr*KLOC*JDDCm	1	2256.87788	2256.87788	2.71	0.0999
QPgmr*FDDDR*IDDCR	1	643.23098	643.23098	0.77	0.3796
FDDDR*IDDCR*JDDCm	1	27.59118	27.59118	0.03	0.8556
QPgm*KLOC*JCoT*JDDCm	1	1068.69462	1068.69462	1.28	0.2575
QPgm*FDDD*MDDC*JDDCm	1	1582.06926	1582.06926	1.90	0.1683

Parameter	Estimate	Standard Error	t Value	Pr > |t|
Intercept	7.973375875	3.15757595	2.53	0.0117
QPgmr	0.479810943	0.04951730	9.69	<.0001
KLOC	-1.386723274	25.37891977	-0.05	0.9564
DDsTim	-0.008317739	0.08287461	-0.10	0.9201
EDRR	0.226689473	0.39552292	0.57	0.5666
FDDDR	-0.034871453	0.11601301	-0.30	0.7638
GCoTim	-0.182261586	0.16214463	-1.12	0.2611
HCRR	-1.513544439	0.85115385	-1.78	0.0755
IDDCR	-0.170372151	0.08846033	-1.93	0.0543
JDDCm	0.131947725	0.04489017	2.94	0.0033
QPgmr*KLOC	-1.151139924	0.49506388	-2.33	0.0202
QPgmr*GCoTim	0.014093616	0.00219346	6.43	<.0001

QPgmr*HCRR	-0.030034076	0.00912683	-3.29	0.0010
QPgmr*IDDCR	-0.004512944	0.00102185	-4.42	<.0001
QPgmr*JDDCm	0.000688872	0.00048120	1.43	0.1524
KLOC*GCoTim	3.171703120	2.11965828	1.50	0.1348
KLOC*HCRR	0.625553189	4.54186925	0.14	0.8905
KLOC*JDDCm	0.636513461	0.57940938	1.10	0.2721
GCoTim*JDDCm	-0.003043093	0.00072396	-4.20	<.0001
HCRR*IDDCR	0.046247340	0.00864724	5.35	<.0001
QPgmr*KLOC*GCoTim	-0.133896891	0.03497419	-3.83	0.0001
QPgmr*KLOC*IDDCR	0.023838214	0.00668162	3.57	0.0004
QPgmr*KLOC*JDDCm	-0.015629781	0.00949492	-1.65	0.0999
QPgmr*FDDDR*IDDCR	0.000029568	0.00003365	0.88	0.3796
FDDDR*IDDCR*JDDCm	-0.000014421	0.00007923	-0.18	0.8556
QPgm*KLOC*JCoT*JDDCm	0.000441760	0.00038999	1.13	0.2575
QPgm*FDDD*MDDC*JDDCm	0.000001175	0.00000085	1.38	0.1683

C.3 COMPILE REGRESSION MODEL FOR (PSPB, C++, OUTLIERS)

The GLM Procedure

Number of observations 920

Dependent Variable: TDDTs DD in Testing (defects/KLOC)

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	9	471233.364	52359.263	65.74	<.0001
Error	910	724775.121	796.456		
Corrected Total	919	1196008.485			

R-Square	Coeff Var	Root MSE	TDDTs Mean
0.394005	100.3984	28.22156	28.10957

Source	DF	Type I SS	Mean Square	F Value	Pr > F
QPgmr	1	238533.8340	238533.8340	299.49	<.0001
KLOC	1	44555.5687	44555.5687	55.94	<.0001
DDsTim	1	7682.2529	7682.2529	9.65	0.0020
EDRR	1	24024.9210	24024.9210	30.16	<.0001
FDDDR	1	13552.0192	13552.0192	17.02	<.0001
GCoTim	1	90794.5485	90794.5485	114.00	<.0001
HCRR	1	190.3839	190.3839	0.24	0.6250
IDDCR	1	43.8565	43.8565	0.06	0.8145
JDDCm	1	51855.9793	51855.9793	65.11	<.0001

Source	DF	Type III SS	Mean Square	F Value	Pr > F
QPgmr	1	107367.6841	107367.6841	134.81	<.0001
KLOC	1	4073.5762	4073.5762	5.11	0.0240

DDsTim	1	121.9735	121.9735	0.15	0.6956
EDRR	1	6291.8791	6291.8791	7.90	0.0051
FDDDR	1	8402.7255	8402.7255	10.55	0.0012
GCoTim	1	54793.2298	54793.2298	68.80	<.0001
HCRR	1	47.6208	47.6208	0.06	0.8069
IDDCR	1	5.6243	5.6243	0.01	0.9330
JDDCm	1	51855.9793	51855.9793	65.11	<.0001

Parameter	Estimate	Standard Error	t Value	Pr > |t|
Intercept	-3.53317911	2.66397493	-1.33	0.1851
QPgmr	0.36097262	0.03108983	11.61	<.0001
KLOC	-22.36150120	9.88767023	-2.26	0.0240
DDsTim	0.04269482	0.10909968	0.39	0.6956
EDRR	-1.89818773	0.67535158	-2.81	0.0051
FDDDR	0.28872659	0.08889097	3.25	0.0012
GCoTim	0.97060218	0.11701964	8.29	<.0001
HCRR	0.15921811	0.65114121	0.24	0.8069
IDDCR	0.00547345	0.06513400	0.08	0.9330
JDDCm	0.16593590	0.02056468	8.07	<.0001

C.4 COMPILE REGRESSION MODEL WITH INTERACTIONS FOR (PSPB, C++, OUTLIERS)

The GLM Procedure

Number of observations 920

Dependent Variable: TDDTs DD in Testing (defects/KLOC)

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	26	682000.223	26230.778	45.57	<.0001
Error	893	514008.262	575.597		
Corrected Total	919	1196008.485			

R-Square	Coeff Var	Root MSE	TDDTs Mean
0.570230	85.35032	23.99161	28.10957

Source	DF	Type I SS	Mean Square	F Value	Pr > F
QPgmr	1	238533.8340	238533.8340	414.41	<.0001
KLOC	1	44555.5687	44555.5687	77.41	<.0001
DDsTim	1	7682.2529	7682.2529	13.35	0.0003
EDRR	1	24024.9210	24024.9210	41.74	<.0001
FDDDR	1	13552.0192	13552.0192	23.54	<.0001
GCoTim	1	90794.5485	90794.5485	157.74	<.0001

HCRR	1	190.3839	190.3839	0.33	0.5654
IDDCR	1	43.8565	43.8565	0.08	0.7826
JDDCm	1	51855.9793	51855.9793	90.09	<.0001
QPgmr*KLOC	1	51322.5655	51322.5655	89.16	<.0001
QPgmr*GCoTim	1	24191.3326	24191.3326	42.03	<.0001
QPgmr*HCRR	1	14897.8947	14897.8947	25.88	<.0001
QPgmr*IDDCR	1	2.0072	2.0072	0.00	0.9529
QPgmr*JDDCm	1	41360.5445	41360.5445	71.86	<.0001
KLOC*GCoTim	1	6761.5254	6761.5254	11.75	0.0006
KLOC*HCRR	1	8995.3028	8995.3028	15.63	<.0001
KLOC*JDDCm	1	475.7257	475.7257	0.83	0.3635
GCoTim*JDDCm	1	37.1526	37.1526	0.06	0.7995
HCRR*IDDCR	1	17943.5591	17943.5591	31.17	<.0001
QPgmr*KLOC*GCoTim	1	12450.4681	12450.4681	21.63	<.0001
QPgmr*KLOC*IDDCR	1	1528.4804	1528.4804	2.66	0.1035
QPgmr*KLOC*JDDCm	1	22181.9771	22181.9771	38.54	<.0001
QPgmr*FDDDR*IDDCR	1	887.7710	887.7710	1.54	0.2146
FDDDR*IDDCR*JDDCm	1	5430.3733	5430.3733	9.43	0.0022
QPgm*KLOC*JCoT*JDDCm	1	2051.6648	2051.6648	3.56	0.0594
QPgm*FDDD*MDDC*JDDCm	1	248.5146	248.5146	0.43	0.5113

Dependent Variable: TDDTs DD in Testing (defects/KLOC)

Source	DF	Type III SS	Mean Square	F Value	Pr > F
QPgmr	1	2.34570	2.34570	0.00	0.9491
KLOC	1	2485.60709	2485.60709	4.32	0.0380
DDsTim	1	559.02517	559.02517	0.97	0.3246
EDRR	1	2397.78772	2397.78772	4.17	0.0415
FDDDR	1	174.11590	174.11590	0.30	0.5825
GCoTim	1	1728.13338	1728.13338	3.00	0.0835
HCRR	1	217.22489	217.22489	0.38	0.5392
IDDCR	1	383.79451	383.79451	0.67	0.4144
JDDCm	1	18262.74520	18262.74520	31.73	<.0001
QPgmr*KLOC	1	798.51238	798.51238	1.39	0.2392
QPgmr*GCoTim	1	475.77621	475.77621	0.83	0.3635
QPgmr*HCRR	1	183.71623	183.71623	0.32	0.5722
QPgmr*IDDCR	1	107.98822	107.98822	0.19	0.6650
QPgmr*JDDCm	1	57317.88688	57317.88688	99.58	<.0001
KLOC*GCoTim	1	354.14953	354.14953	0.62	0.4330
KLOC*HCRR	1	193.74390	193.74390	0.34	0.5619
KLOC*JDDCm	1	14494.73587	14494.73587	25.18	<.0001
GCoTim*JDDCm	1	951.51233	951.51233	1.65	0.1989
HCRR*IDDCR	1	4435.95996	4435.95996	7.71	0.0056
QPgmr*KLOC*GCoTim	1	98.68005	98.68005	0.17	0.6789
QPgmr*KLOC*IDDCR	1	27.96249	27.96249	0.05	0.8256
QPgmr*KLOC*JDDCm	1	18259.85940	18259.85940	31.72	<.0001
QPgmr*FDDDR*IDDCR	1	6218.99232	6218.99232	10.80	0.0011
FDDDR*IDDCR*JDDCm	1	1256.57645	1256.57645	2.18	0.1399
QPgm*KLOC*JCoT*JDDCm	1	2144.73506	2144.73506	3.73	0.0539
QPgm*FDDD*MDDC*JDDCm	1	248.51456	248.51456	0.43	0.5113

Parameter	Estimate	Standard Error	t Value	Pr > |t|
Intercept	20.98140979	4.51893343	4.64	<.0001
QPgmr	0.00565435	0.08857398	0.06	0.9491

KLOC	-53.16460241	25.58383536	-2.08	0.0380
DDsTim	0.09573212	0.09714072	0.99	0.3246
EDRR	-1.20173555	0.58879331	-2.04	0.0415
FDDDR	0.05746353	0.10447982	0.55	0.5825
GCoTim	0.47640336	0.27494480	1.73	0.0835
HCRR	-0.64644791	1.05229635	-0.61	0.5392
IDDCR	-0.10499646	0.12858331	-0.82	0.4144
JDDCm	-0.33948120	0.06026873	-5.63	<.0001
QPgmr*KLOC	0.64567584	0.54819243	1.18	0.2392
QPgmr*GCoTim	0.00324878	0.00357338	0.91	0.3635
QPgmr*HCRR	0.01024159	0.01812812	0.56	0.5722
QPgmr*IDDCR	-0.00136498	0.00315136	-0.43	0.6650
QPgmr*JDDCm	0.00870856	0.00087269	9.98	<.0001
KLOC*GCoTim	-2.07961061	2.65123370	-0.78	0.4330
KLOC*HCRR	-3.39898255	5.85860518	-0.58	0.5619
KLOC*JDDCm	2.36778475	0.47184177	5.02	<.0001
GCoTim*JDDCm	0.00179134	0.00139325	1.29	0.1989
HCRR*IDDCR	0.02072646	0.00746605	2.78	0.0056
QPgmr*KLOC*GCoTim	0.02114498	0.05106837	0.41	0.6789
QPgmr*KLOC*IDDCR	-0.00274769	0.01246632	-0.22	0.8256
QPgmr*KLOC*JDDCm	-0.04197092	0.00745176	-5.63	<.0001
QPgmr*FDDDR*IDDCR	-0.00008755	0.00002664	-3.29	0.0011
FDDDR*IDDCR*JDDCm	0.00005846	0.00003957	1.48	0.1399
QPgm*KLOC*JCoT*JDDCm	-0.00055831	0.00028923	-1.93	0.0539
QPgm*FDDD*MDDC*JDDCm	0.00000053	0.00000080	0.66	0.5113

C.5 COMPILE REGRESSION MODEL FOR (PSPB, C, NOOUTLIERS)

The GLM Procedure

Number of observations 1711

Dependent Variable: TDDTs DD in Testing (defects/KLOC)

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	9	452568.423	50285.380	98.01	<.0001
Error	1701	872722.394	513.064		
Corrected Total	1710	1325290.817			

R-Square	Coeff Var	Root MSE	TDDTs Mean
0.341486	80.88062	22.65092	28.00538

Source	DF	Type I SS	Mean Square	F Value	Pr > F
PgmrAb	1	176181.3310	176181.3310	343.39	<.0001
KLOC	1	74495.9772	74495.9772	145.20	<.0001
DDsTim	1	6.9931	6.9931	0.01	0.9071
EDRR	1	69821.8671	69821.8671	136.09	<.0001

FDDDR	1	166.7246	166.7246	0.32	0.5687
GCoTim	1	29555.7466	29555.7466	57.61	<.0001
HCCR	1	49062.0844	49062.0844	95.63	<.0001
IDDCR	1	4691.2356	4691.2356	9.14	0.0025
JDDCm	1	48586.4638	48586.4638	94.70	<.0001

Source	DF	Type III SS	Mean Square	F Value	Pr > F
PgmrAb	1	133071.1807	133071.1807	259.37	<.0001
KLOC	1	23978.0737	23978.0737	46.74	<.0001
DDsTim	1	1421.1501	1421.1501	2.77	0.0962
EDRR	1	179.4885	179.4885	0.35	0.5543
FDDDR	1	1256.3482	1256.3482	2.45	0.1178
GCoTim	1	8388.2222	8388.2222	16.35	<.0001
HCCR	1	16368.1461	16368.1461	31.90	<.0001
IDDCR	1	3490.9815	3490.9815	6.80	0.0092
JDDCm	1	48586.4638	48586.4638	94.70	<.0001

Parameter	Estimate	Standard Error	t Value	Pr > |t|
Intercept	17.82635277	1.56887991	11.36	<.0001
PgmrAb	0.23530681	0.01461095	16.10	<.0001
KLOC	-58.38809583	8.54088886	-6.84	<.0001
DDsTim	0.10919400	0.06560920	1.66	0.0962
EDRR	-0.18248740	0.30853214	-0.59	0.5543
FDDDR	0.09533294	0.06092196	1.56	0.1178
GCoTim	0.20144163	0.04981959	4.04	<.0001
HCCR	-2.03515964	0.36031691	-5.65	<.0001
IDDCR	-0.10433942	0.04000005	-2.61	0.0092
JDDCm	0.11376523	0.01169063	9.73	<.0001

C.6 COMPILE REGRESSION MODEL WITH INTERACTIONS FOR (PSPB, C, NOOUTLIERS)

The GLM Procedure

Number of observations 1705

Dependent Variable: TDDTs DD in Testing (defects/KLOC)

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	26	544743.862	20951.687	45.39	<.0001
Error	1678	774573.004	461.605		
Corrected Total	1704	1319316.866			

R-Square	Coeff Var	Root MSE	TDDTs Mean
0.412898	77.20018	21.48499	27.83024

Source	DF	Type I SS	Mean Square	F Value	Pr > F
PgmrAb	1	185425.4955	185425.4955	401.70	<.0001
KLOC	1	61462.2165	61462.2165	133.15	<.0001
DDsTim	1	161.4403	161.4403	0.35	0.5543
EDRR	1	80757.9448	80757.9448	174.95	<.0001
FDDDR	1	31.8665	31.8665	0.07	0.7928
GCoTim	1	36930.8677	36930.8677	80.01	<.0001
HCCR	1	33302.2912	33302.2912	72.14	<.0001
IDDCR	1	1958.1438	1958.1438	4.24	0.0396
JDDCm	1	47694.7709	47694.7709	103.32	<.0001
PgmrAb*KLOC	1	29181.4638	29181.4638	63.22	<.0001
PgmrAb*GCoTim	1	14667.0393	14667.0393	31.77	<.0001
PgmrAb*HCCR	1	13804.2840	13804.2840	29.90	<.0001
PgmrAb*IDDCR	1	827.4200	827.4200	1.79	0.1808
PgmrAb*JDDCm	1	528.7466	528.7466	1.15	0.2847
KLOC*GCoTim	1	3.1720	3.1720	0.01	0.9339
KLOC*HCCR	1	111.7268	111.7268	0.24	0.6228
KLOC*JDDCm	1	4266.1335	4266.1335	9.24	0.0024
GCoTim*JDDCm	1	3671.6585	3671.6585	7.95	0.0049
HCCR*IDDCR	1	16684.6168	16684.6168	36.14	<.0001
PgmrAb*KLOC*GCoTim	1	8020.8651	8020.8651	17.38	<.0001
PgmrAb*KLOC*IDDCR	1	4052.7452	4052.7452	8.78	0.0031
PgmrAb*KLOC*JDDCm	1	14.3783	14.3783	0.03	0.8599
PgmrAb*FDDDR*IDDCR	1	349.3723	349.3723	0.76	0.3844
FDDDR*IDDCR*JDDCm	1	276.3257	276.3257	0.60	0.4392
Pgmr*KLOC*GCoT*JDDCm	1	55.0625	55.0625	0.12	0.7299
Pgmr*FDDD*IDDC*JDDCm	1	503.8143	503.8143	1.09	0.2963

Source	DF	Type III SS	Mean Square	F Value	Pr > F
PgmrAb	1	14847.13966	14847.13966	32.16	<.0001
KLOC	1	1155.56996	1155.56996	2.50	0.1138
DDsTim	1	3.05958	3.05958	0.01	0.9351
EDRR	1	5.52283	5.52283	0.01	0.9129
FDDDR	1	276.27145	276.27145	0.60	0.4393
GCoTim	1	910.88752	910.88752	1.97	0.1603
HCCR	1	1866.36628	1866.36628	4.04	0.0445
IDDCR	1	2194.31261	2194.31261	4.75	0.0294
JDDCm	1	1989.47511	1989.47511	4.31	0.0380
PgmrAb*KLOC	1	662.59961	662.59961	1.44	0.2311
PgmrAb*GCoTim	1	14606.61658	14606.61658	31.64	<.0001
PgmrAb*HCCR	1	4698.93910	4698.93910	10.18	0.0014
PgmrAb*IDDCR	1	3368.61510	3368.61510	7.30	0.0070
PgmrAb*JDDCm	1	159.34576	159.34576	0.35	0.5569
KLOC*GCoTim	1	1785.99491	1785.99491	3.87	0.0493
KLOC*HCCR	1	1.57097	1.57097	0.00	0.9535
KLOC*JDDCm	1	714.90053	714.90053	1.55	0.2135
GCoTim*JDDCm	1	1261.81913	1261.81913	2.73	0.0984
HCCR*IDDCR	1	16403.77219	16403.77219	35.54	<.0001
PgmrAb*KLOC*GCoTim	1	4665.06107	4665.06107	10.11	0.0015
PgmrAb*KLOC*IDDCR	1	3080.17002	3080.17002	6.67	0.0099
PgmrAb*KLOC*JDDCm	1	11.96236	11.96236	0.03	0.8721
PgmrAb*FDDDR*IDDCR	1	132.32336	132.32336	0.29	0.5924
FDDDR*IDDCR*JDDCm	1	786.41823	786.41823	1.70	0.1920
Pgmr*KLOC*GCoT*JDDCm	1	43.13441	43.13441	0.09	0.7599

Pgmr*FDDD*IDDC*JDDCm	1	503.81430	503.81430	1.09	0.2963

Parameter	Estimate	Standard Error	t Value	Pr > |t|
Intercept	15.83744334	2.78945475	5.68	<.0001
PgmrAb	0.27624909	0.04870961	5.67	<.0001
KLOC	-33.84707782	21.39233836	-1.58	0.1138
DDsTim	0.00522418	0.06416854	0.08	0.9351
EDRR	-0.03722814	0.34035006	-0.11	0.9129
FDDDR	0.07273790	0.09402167	0.77	0.4393
GCoTim	-0.20389290	0.14514595	-1.40	0.1603
HCCR	-1.44779065	0.72001663	-2.01	0.0445
IDDCR	-0.16878653	0.07741475	-2.18	0.0294
JDDCm	0.09298733	0.04479089	2.08	0.0380
PgmrAb*KLOC	-0.57525338	0.48014100	-1.20	0.2311
PgmrAb*GCoTim	0.01382192	0.00245713	5.63	<.0001
PgmrAb*HCCR	-0.02929011	0.00918029	-3.19	0.0014
PgmrAb*IDDCR	-0.00306803	0.00113571	-2.70	0.0070
PgmrAb*JDDCm	0.00025176	0.00042850	0.59	0.5569
KLOC*GCoTim	3.62618867	1.84350996	1.97	0.0493
KLOC*HCCR	0.22938806	3.93207331	0.06	0.9535
KLOC*JDDCm	0.60646029	0.48732059	1.24	0.2135
GCoTim*JDDCm	-0.00181884	0.00110009	-1.65	0.0984
HCCR*IDDCR	0.04042283	0.00678094	5.96	<.0001
PgmrAb*KLOC*GCoTim	-0.12188286	0.03833974	-3.18	0.0015
PgmrAb*KLOC*IDDCR	0.02240647	0.00867404	2.58	0.0099
PgmrAb*KLOC*JDDCm	-0.00146798	0.00911897	-0.16	0.8721
PgmrAb*FDDDR*IDDCR	0.00001472	0.00002749	0.54	0.5924
FDDDR*IDDCR*JDDCm	-0.00013476	0.00010325	-1.31	0.1920
Pgmr*KLOC*GCoT*JDDCm	-0.00012705	0.00041562	-0.31	0.7599
Pgmr*FDDD*IDDC*JDDCm	0.00000166	0.00000158	1.04	0.2963

C.7 COMPILE REGRESSION MODEL FOR (PSPB, C++, NOOUTLIERS)

The GLM Procedure

Number of observations 892

Dependent Variable: TDDTs DD in Testing (defects/KLOC)

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	9	166539.5510	18504.3946	42.00	<.0001
Error	882	388579.1474	440.5659		
Corrected Total	891	555118.6984			

R-Square	Coeff Var	Root MSE	TDDTs Mean
0.300007	86.48124	20.98966	24.27077

Source	DF	Type I SS	Mean Square	F Value	Pr > F
PgmrAb	1	59813.03423	59813.03423	135.76	<.0001
KLOC	1	22833.39692	22833.39692	51.83	<.0001
DDsTim	1	669.41692	669.41692	1.52	0.2180
EDRR	1	43236.50622	43236.50622	98.14	<.0001
FDDDR	1	30.36474	30.36474	0.07	0.7930
GCoTim	1	26002.84200	26002.84200	59.02	<.0001
HCCR	1	3557.68114	3557.68114	8.08	0.0046
IDDCR	1	185.92645	185.92645	0.42	0.5161
JDDCm	1	10210.38237	10210.38237	23.18	<.0001

Source	DF	Type III SS	Mean Square	F Value	Pr > F
PgmrAb	1	38822.38725	38822.38725	88.12	<.0001
KLOC	1	4970.27357	4970.27357	11.28	0.0008
DDsTim	1	364.96204	364.96204	0.83	0.3630
EDRR	1	3153.10340	3153.10340	7.16	0.0076
FDDDR	1	309.97469	309.97469	0.70	0.4018
GCoTim	1	18738.77161	18738.77161	42.53	<.0001
HCCR	1	778.22104	778.22104	1.77	0.1842
IDDCR	1	222.10555	222.10555	0.50	0.4779
JDDCm	1	10210.38237	10210.38237	23.18	<.0001

Parameter	Estimate	Standard Error	t Value	Pr > |t|
Intercept	9.82921216	2.11768433	4.64	<.0001
PgmrAb	0.23336474	0.02485991	9.39	<.0001
KLOC	-25.71253595	7.65526480	-3.36	0.0008
DDsTim	0.08094949	0.08893971	0.91	0.3630
EDRR	-1.58618880	0.59291334	-2.68	0.0076
FDDDR	0.07282847	0.08682475	0.84	0.4018
GCoTim	0.60775135	0.09318818	6.52	<.0001
HCCR	-0.72865909	0.54824964	-1.33	0.1842
IDDCR	-0.03861900	0.05439096	-0.71	0.4779
JDDCm	0.08032920	0.01668622	4.81	<.0001

C.8 COMPILE REGRESSION MODEL WITH INTERACTIONS FOR (PSPB, C++, NOOUTLIERS)

The GLM Procedure

Number of observations 884

Dependent Variable: TDDTs DD in Testing (defects/KLOC)

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	26	291356.2288	11206.0088	28.51	<.0001

Error	857	336902.0468	393.1179	
Corrected Total	883	628258.2756		

R-Square	Coeff Var	Root MSE	TDDTs Mean
0.463752	81.33140	19.82720	24.37829

Source	DF	Type I SS	Mean Square	F Value	Pr > F
PgmrAb	1	104113.3820	104113.3820	264.84	<.0001
KLOC	1	23194.4445	23194.4445	59.00	<.0001
DDsTim	1	48.0820	48.0820	0.12	0.7266
EDRR	1	35159.7892	35159.7892	89.44	<.0001
FDDDR	1	579.4338	579.4338	1.47	0.2251
GCoTim	1	28544.7235	28544.7235	72.61	<.0001
HCCR	1	3539.5178	3539.5178	9.00	0.0028
IDDCR	1	45.3275	45.3275	0.12	0.7343
JDDCm	1	16371.5002	16371.5002	41.65	<.0001
PgmrAb*KLOC	1	11464.4237	11464.4237	29.16	<.0001
PgmrAb*GCoTim	1	23023.9475	23023.9475	58.57	<.0001
PgmrAb*HCCR	1	3665.4414	3665.4414	9.32	0.0023
PgmrAb*IDDCR	1	292.3120	292.3120	0.74	0.3888
PgmrAb*JDDCm	1	11335.6172	11335.6172	28.84	<.0001
KLOC*GCoTim	1	20.4778	20.4778	0.05	0.8195
KLOC*HCCR	1	1409.1021	1409.1021	3.58	0.0587
KLOC*JDDCm	1	630.0462	630.0462	1.60	0.2059
GCoTim*JDDCm	1	3393.8182	3393.8182	8.63	0.0034
HCCR*IDDCR	1	5137.2858	5137.2858	13.07	0.0003
PgmrAb*KLOC*GCoTim	1	14556.7058	14556.7058	37.03	<.0001
PgmrAb*KLOC*IDDCR	1	276.8374	276.8374	0.70	0.4016
PgmrAb*KLOC*JDDCm	1	1425.5066	1425.5066	3.63	0.0572
PgmrAb*FDDDR*IDDCR	1	17.3264	17.3264	0.04	0.8338
FDDDR*IDDCR*JDDCm	1	1162.2406	1162.2406	2.96	0.0859
Pgmr*KLOC*GCoT*JDDCm	1	17.0194	17.0194	0.04	0.8352
Pgmr*FDDD*IDDC*JDDCm	1	1931.9201	1931.9201	4.91	0.0269

Source	DF	Type III SS	Mean Square	F Value	Pr > F
PgmrAb	1	66.83404	66.83404	0.17	0.6802
KLOC	1	8367.92833	8367.92833	21.29	<.0001
DDsTim	1	306.58851	306.58851	0.78	0.3774
EDRR	1	982.62619	982.62619	2.50	0.1142
FDDDR	1	0.26383	0.26383	0.00	0.9793
GCoTim	1	393.04726	393.04726	1.00	0.3176
HCCR	1	1155.18863	1155.18863	2.94	0.0869
IDDCR	1	872.49526	872.49526	2.22	0.1367
JDDCm	1	79.19711	79.19711	0.20	0.6537
PgmrAb*KLOC	1	5723.17336	5723.17336	14.56	0.0001
PgmrAb*GCoTim	1	13277.28025	13277.28025	33.77	<.0001
PgmrAb*HCCR	1	290.58441	290.58441	0.74	0.3902
PgmrAb*IDDCR	1	105.87516	105.87516	0.27	0.6039
PgmrAb*JDDCm	1	5924.65508	5924.65508	15.07	0.0001
KLOC*GCoTim	1	3797.69892	3797.69892	9.66	0.0019
KLOC*HCCR	1	605.02487	605.02487	1.54	0.2151
KLOC*JDDCm	1	1734.51733	1734.51733	4.41	0.0360
GCoTim*JDDCm	1	3327.95105	3327.95105	8.47	0.0037

HCCR*IDDCR	1	3918.57360	3918.57360	9.97	0.0016
PgmrAb*KLOC*GCoTim	1	8390.17752	8390.17752	21.34	<.0001
PgmrAb*KLOC*IDDCR	1	178.56869	178.56869	0.45	0.5005
PgmrAb*KLOC*JDDCm	1	1273.15584	1273.15584	3.24	0.0723
PgmrAb*FDDDR*IDDCR	1	39.01813	39.01813	0.10	0.7528
FDDDR*IDDCR*JDDCm	1	2745.96740	2745.96740	6.99	0.0084
Pgmr*KLOC*GCoT*JDDCm	1	11.41940	11.41940	0.03	0.8647
Pgmr*FDDD*IDDC*JDDCm	1	1931.92013	1931.92013	4.91	0.0269

Parameter	Estimate	Standard Error	t Value	Pr > |t|
Intercept	22.3530181	4.21267417	5.31	<.0001
PgmrAb	-0.0342946	0.08317424	-0.41	0.6802
KLOC	-111.5722083	24.18290483	-4.61	<.0001
DDsTim	0.0794410	0.08995561	0.88	0.3774
EDRR	-0.8336760	0.52730820	-1.58	0.1142
FDDDR	-0.0028817	0.11123687	-0.03	0.9793
GCoTim	-0.2815211	0.28154644	-1.00	0.3176
HCCR	-1.7026238	0.99323876	-1.71	0.0869
IDDCR	-0.1791328	0.12024167	-1.49	0.1367
JDDCm	-0.0294056	0.06551438	-0.45	0.6537
PgmrAb*KLOC	1.9631478	0.51451253	3.82	0.0001
PgmrAb*GCoTim	0.0241036	0.00414753	5.81	<.0001
PgmrAb*HCCR	-0.0146178	0.01700232	-0.86	0.3902
PgmrAb*IDDCR	0.0015358	0.00295939	0.52	0.6039
PgmrAb*JDDCm	0.0038903	0.00100211	3.88	0.0001
KLOC*GCoTim	8.1514228	2.62261517	3.11	0.0019
KLOC*HCCR	7.0997841	5.72294990	1.24	0.2151
KLOC*JDDCm	1.2185559	0.58011962	2.10	0.0360
GCoTim*JDDCm	-0.0061636	0.00211841	-2.91	0.0037
HCCR*IDDCR	0.0212104	0.00671810	3.16	0.0016
PgmrAb*KLOC*GCoTim	-0.2424967	0.05249061	-4.62	<.0001
PgmrAb*KLOC*IDDCR	-0.0112783	0.01673406	-0.67	0.5005
PgmrAb*KLOC*JDDCm	-0.0238445	0.01324976	-1.80	0.0723
PgmrAb*FDDDR*IDDCR	0.0000178	0.00005653	0.32	0.7528
FDDDR*IDDCR*JDDCm	0.0002259	0.00008548	2.64	0.0084
Pgmr*KLOC*GCoT*JDDCm	0.0000893	0.00052420	0.17	0.8647
Pgmr*FDDD*IDDC*JDDCm	-0.0000047	0.00000210	-2.22	0.0269

C.9 COMPILE MIXED MODELS FOR (PSPB, C, OUTLIERS)

The Mixed Procedure

Dimensions

Covariance Parameters	55
Columns in X	9
Columns in Z	0
Subjects	197
Max Obs Per Subject	10
Observations Used	1758

Observations Not Used	0
Total Observations	1758

Iteration History

Iteration	Evaluations	-2 Res Log Like	Criterion
0	1	6054.96563064	
1	2	5738.13230271	0.00060339
2	1	5737.30657788	0.00001357
3	1	5737.28912193	0.00000001
4	1	5737.28910593	0.00000000

Convergence criteria met.

Estimated R Matrix for Student 1995m10

Row	Col1	Col2	Col3	Col4	Col5	Col6	Col7
1	2.1682	0.5629	0.4006	0.3713	0.4196	0.3628	0.3058
2	0.5629	2.0100	0.5104	0.7049	0.5652	0.3164	0.3287
3	0.4006	0.5104	1.3926	0.4115	0.2034	0.2168	0.1530
4	0.3713	0.7049	0.4115	2.4451	0.5020	0.3014	0.2416
5	0.4196	0.5652	0.2034	0.5020	1.6398	0.1888	0.3959
6	0.3628	0.3164	0.2168	0.3014	0.1888	0.8740	0.3197
7	0.3058	0.3287	0.1530	0.2416	0.3959	0.3197	2.2193

Estimated R Correlation Matrix for Student 1995m10

Row	Col1	Col2	Col3	Col4	Col5	Col6	Col7
1	1.0000	0.2696	0.2305	0.1613	0.2225	0.2636	0.1394
2	0.2696	1.0000	0.3051	0.3180	0.3113	0.2387	0.1556
3	0.2305	0.3051	1.0000	0.2230	0.1346	0.1965	0.08703
4	0.1613	0.3180	0.2230	1.0000	0.2507	0.2062	0.1037
5	0.2225	0.3113	0.1346	0.2507	1.0000	0.1577	0.2076
6	0.2636	0.2387	0.1965	0.2062	0.1577	1.0000	0.2295
7	0.1394	0.1556	0.08703	0.1037	0.2076	0.2295	1.0000

Covariance Parameter Estimates

Cov Parm	Subject	Estimate
UN(1,1)	Student	2.1682
UN(2,1)	Student	0.5629
UN(2,2)	Student	2.0100
UN(3,1)	Student	0.4006
UN(3,2)	Student	0.5104
UN(3,3)	Student	1.3926
UN(4,1)	Student	0.3713
UN(4,2)	Student	0.7049
UN(4,3)	Student	0.4115
UN(4,4)	Student	2.4451
UN(5,1)	Student	0.4196
UN(5,2)	Student	0.5652
UN(5,3)	Student	0.2034
UN(5,4)	Student	0.5020

UN(5,5)	Student	1.6398
UN(6,1)	Student	0.3628
UN(6,2)	Student	0.3164
UN(6,3)	Student	0.2168
UN(6,4)	Student	0.3014
UN(6,5)	Student	0.1888
UN(6,6)	Student	0.8740
UN(7,1)	Student	0.3058
UN(7,2)	Student	0.3287
UN(7,3)	Student	0.1530
UN(7,4)	Student	0.2416
UN(7,5)	Student	0.3959
UN(7,6)	Student	0.3197
UN(7,7)	Student	2.2193
UN(8,1)	Student	0.1732
UN(8,2)	Student	0.06235
UN(8,3)	Student	0.1473
UN(8,4)	Student	0.4334
UN(8,5)	Student	0.09947
UN(8,6)	Student	0.1502
UN(8,7)	Student	0.5163
UN(8,8)	Student	2.3354
UN(9,1)	Student	0.4035
UN(9,2)	Student	0.05281
UN(9,3)	Student	0.3657
UN(9,4)	Student	0.09652
UN(9,5)	Student	0.2284
UN(9,6)	Student	0.5841
UN(9,7)	Student	0.5218
UN(9,8)	Student	0.5944
UN(9,9)	Student	1.6218
UN(10,1)	Student	0.4895
UN(10,2)	Student	0.08354
UN(10,3)	Student	0.1559
UN(10,4)	Student	0.3676
UN(10,5)	Student	0.2044
UN(10,6)	Student	0.4573
UN(10,7)	Student	0.4787
UN(10,8)	Student	0.3149
UN(10,9)	Student	0.5715
UN(10,10)	Student	1.3903

Fit Statistics

-2 Res Log Likelihood	5737.3
AIC (smaller is better)	5847.3
AICC (smaller is better)	5850.9
BIC (smaller is better)	6027.9

Null Model Likelihood Ratio Test

DF	Chi-Square	Pr > ChiSq
54	317.68	<.0001

Solution for Fixed Effects

Effect	Estimate	Standard Error	DF	t Value	Pr > |t|
Intercept	1.0056	0.1789	880	5.62	<.0001
KLnKLOC	-0.05516	0.05128	731	-1.08	0.2824
DLnDsTim	0.1556	0.04503	947	3.46	0.0006
ELnDRR	-0.09745	0.06437	798	-1.51	0.1305
FLnDDDR	0.1226	0.03751	584	3.27	0.0011
GLnCoTim	0.2766	0.06317	1002	4.38	<.0001
HLnCRR	-0.1935	0.06307	796	-3.07	0.0022
ILnDDCR	-0.00297	0.03888	613	-0.08	0.9391
JLnDDCm	0.1063	0.02273	1494	4.68	<.0001

Type 3 Tests of Fixed Effects

Effect	Num DF	Den DF	F Value	Pr > F
KLnKLOC	1	731	1.16	0.2824
DLnDsTim	1	947	11.94	0.0006
ELnDRR	1	798	2.29	0.1305
FLnDDDR	1	584	10.68	0.0011
GLnCoTim	1	1002	19.17	<.0001
HLnCRR	1	796	9.41	0.0022
ILnDDCR	1	613	0.01	0.9391
JLnDDCm	1	1494	21.87	<.0001

C.10 COMPILE MIXED MODELS WITH INTERACTIONS FOR (PSPB, C, OUTLIERS)

The Mixed Procedure

Dimensions

Covariance Parameters	55
Columns in X	15
Columns in Z	0
Subjects	197
Max Obs Per Subject	10
Observations Used	1758
Observations Not Used	0
Total Observations	1758

Iteration History

Iteration	Evaluations	-2 Res Log Like	Criterion
0	1	6054.83902591	
1	2	5731.82809085	0.00101888
2	1	5730.39527631	0.00004449

3	1	5730.33699984	0.00000015
4	1	5730.33680973	0.00000000

Convergence criteria met.

Estimated R Matrix for Student 1995m10

Row	Col1	Col2	Col3	Col4	Col5	Col6	Col7
1	2.1373	0.5550	0.4163	0.3732	0.4358	0.3602	0.3421
2	0.5550	2.0284	0.4974	0.7213	0.5793	0.3225	0.3320
3	0.4163	0.4974	1.3148	0.3689	0.2319	0.2284	0.1491
4	0.3732	0.7213	0.3689	2.4432	0.5253	0.3254	0.2952
5	0.4358	0.5793	0.2319	0.5253	1.6055	0.1945	0.4022
6	0.3602	0.3225	0.2284	0.3254	0.1945	0.8514	0.2894
7	0.3421	0.3320	0.1491	0.2952	0.4022	0.2894	2.2507

Estimated R Correlation Matrix for Student 1995m10

Row	Col1	Col2	Col3	Col4	Col5	Col6	Col7
1	1.0000	0.2665	0.2483	0.1633	0.2353	0.2670	0.1560
2	0.2665	1.0000	0.3046	0.3240	0.3210	0.2454	0.1554
3	0.2483	0.3046	1.0000	0.2058	0.1596	0.2159	0.08669
4	0.1633	0.3240	0.2058	1.0000	0.2652	0.2256	0.1259
5	0.2353	0.3210	0.1596	0.2652	1.0000	0.1663	0.2116
6	0.2670	0.2454	0.2159	0.2256	0.1663	1.0000	0.2090
7	0.1560	0.1554	0.08669	0.1259	0.2116	0.2090	1.0000

Covariance Parameter Estimates

Cov Parm	Subject	Estimate
UN(1,1)	Student	2.1373
UN(2,1)	Student	0.5550
UN(2,2)	Student	2.0284
UN(3,1)	Student	0.4163
UN(3,2)	Student	0.4974
UN(3,3)	Student	1.3148
UN(4,1)	Student	0.3732
UN(4,2)	Student	0.7213
UN(4,3)	Student	0.3689
UN(4,4)	Student	2.4432
UN(5,1)	Student	0.4358
UN(5,2)	Student	0.5793
UN(5,3)	Student	0.2319
UN(5,4)	Student	0.5253
UN(5,5)	Student	1.6055
UN(6,1)	Student	0.3602
UN(6,2)	Student	0.3225
UN(6,3)	Student	0.2284
UN(6,4)	Student	0.3254
UN(6,5)	Student	0.1945
UN(6,6)	Student	0.8514
UN(7,1)	Student	0.3421
UN(7,2)	Student	0.3320
UN(7,3)	Student	0.1491

UN(7,4)	Student	0.2952
UN(7,5)	Student	0.4022
UN(7,6)	Student	0.2894
UN(7,7)	Student	2.2507
UN(8,1)	Student	0.2295
UN(8,2)	Student	0.04714
UN(8,3)	Student	0.1716
UN(8,4)	Student	0.4935
UN(8,5)	Student	0.1099
UN(8,6)	Student	0.1350
UN(8,7)	Student	0.4639
UN(8,8)	Student	2.3588
UN(9,1)	Student	0.3710
UN(9,2)	Student	0.03693
UN(9,3)	Student	0.3859
UN(9,4)	Student	0.06697
UN(9,5)	Student	0.2244
UN(9,6)	Student	0.5248
UN(9,7)	Student	0.4552
UN(9,8)	Student	0.5941
UN(9,9)	Student	1.5354
UN(10,1)	Student	0.4509
UN(10,2)	Student	0.09155
UN(10,3)	Student	0.1330
UN(10,4)	Student	0.3165
UN(10,5)	Student	0.1958
UN(10,6)	Student	0.4833
UN(10,7)	Student	0.5097
UN(10,8)	Student	0.3219
UN(10,9)	Student	0.5596
UN(10,10)	Student	1.3669

Fit Statistics

-2 Res Log Likelihood	5730.3
AIC (smaller is better)	5840.3
AICC (smaller is better)	5844.0
BIC (smaller is better)	6020.9

Null Model Likelihood Ratio Test

DF	Chi-Square	Pr > ChiSq
54	324.50	<.0001

Solution for Fixed Effects

Effect	Estimate	Standard Error	DF	t Value	Pr > |t|
Intercept	1.4351	0.4364	933	3.29	0.0010
KLnKLOC	-0.2160	0.1462	996	-1.48	0.1398
DLnDsTim	-0.02944	0.1396	813	-0.21	0.8331
ELnDRR	-0.3930	0.1268	575	-3.10	0.0020
FLnDDDR	-0.1812	0.1280	451	-1.42	0.1576
GLnCoTim	-0.05378	0.1221	1097	-0.44	0.6596
HLnCRR	-0.2407	0.06493	788	-3.71	0.0002

ILnDDCR	0.5272	0.1230	531	4.29	<.0001
JLnDDCm	-0.1168	0.08290	1229	-1.41	0.1589
KLnKLOC*DLnDsTim	-0.06684	0.05204	945	-1.28	0.1993
KLnKLOC*ELnDRR	-0.1322	0.04389	623	-3.01	0.0027
KLnKLOC*FLnDDDR	-0.1323	0.05269	527	-2.51	0.0123
KLnKLOC*ILnDDCR	0.2475	0.05050	721	4.90	<.0001
GLnCoTim*JLnDDCm	0.08671	0.03190	1128	2.72	0.0067
KLnK*GLnC*HLnC*ILnDD	-0.01066	0.003645	664	-2.93	0.0036

Type 3 Tests of Fixed Effects

Effect	Num DF	Den DF	F Value	Pr > F
KLnKLOC	1	996	2.18	0.1398
DLnDsTim	1	813	0.04	0.8331
ELnDRR	1	575	9.60	0.0020
FLnDDDR	1	451	2.00	0.1576
GLnCoTim	1	1097	0.19	0.6596
HLnCRR	1	788	13.74	0.0002
ILnDDCR	1	531	18.37	<.0001
JLnDDCm	1	1229	1.99	0.1589
KLnKLOC*DLnDsTim	1	945	1.65	0.1993
KLnKLOC*ELnDRR	1	623	9.07	0.0027
KLnKLOC*FLnDDDR	1	527	6.30	0.0123
KLnKLOC*ILnDDCR	1	721	24.02	<.0001
GLnCoTim*JLnDDCm	1	1128	7.39	0.0067
KLnK*GLnC*HLnC*ILnDD	1	664	8.56	0.0036

C.11 COMPILE MIXED MODELS WITH INTERACTIONS FOR (PSPB, C, NOOUTLIERS)

The Mixed Procedure

Dimensions

Covariance Parameters	55
Columns in X	9
Columns in Z	0
Subjects	197
Max Obs Per Subject	10
Observations Used	1711
Observations Not Used	0
Total Observations	1711

Iteration History

Iteration	Evaluations	-2 Res Log Like	Criterion
0	1	5839.53298935	
1	2	5536.64757938	0.00102614
2	1	5535.28486200	0.00003602

3	1	5535.24020277	0.00000008
4	1	5535.24010328	0.00000000

Convergence criteria met.

Estimated R Matrix for Student 1995m10

Row	Col1	Col2	Col3	Col4	Col5	Col6	Col7
1	2.1033	0.5066	0.3211	0.3134	0.3842	0.3729	0.3071
2	0.5066	1.9528	0.3270	0.7759	0.5939	0.3554	0.3479
3	0.3211	0.3270	1.1972	0.2729	0.2339	0.1394	0.02399
4	0.3134	0.7759	0.2729	2.4203	0.5248	0.3515	0.1612
5	0.3842	0.5939	0.2339	0.5248	1.6156	0.1965	0.3254
6	0.3729	0.3554	0.1394	0.3515	0.1965	0.8853	0.3075
7	0.3071	0.3479	0.02399	0.1612	0.3254	0.3075	2.1271

Estimated R Correlation Matrix for Student 1995m10

Row	Col1	Col2	Col3	Col4	Col5	Col6	Col7
1	1.0000	0.2500	0.2023	0.1389	0.2084	0.2733	0.1452
2	0.2500	1.0000	0.2139	0.3569	0.3344	0.2703	0.1707
3	0.2023	0.2139	1.0000	0.1603	0.1682	0.1354	0.01504
4	0.1389	0.3569	0.1603	1.0000	0.2654	0.2401	0.07106
5	0.2084	0.3344	0.1682	0.2654	1.0000	0.1643	0.1755
6	0.2733	0.2703	0.1354	0.2401	0.1643	1.0000	0.2241
7	0.1452	0.1707	0.01504	0.07106	0.1755	0.2241	1.0000

Covariance Parameter Estimates

Cov Parm	Subject	Estimate
UN(1,1)	Student	2.1033
UN(2,1)	Student	0.5066
UN(2,2)	Student	1.9528
UN(3,1)	Student	0.3211
UN(3,2)	Student	0.3270
UN(3,3)	Student	1.1972
UN(4,1)	Student	0.3134
UN(4,2)	Student	0.7759
UN(4,3)	Student	0.2729
UN(4,4)	Student	2.4203
UN(5,1)	Student	0.3842
UN(5,2)	Student	0.5939
UN(5,3)	Student	0.2339
UN(5,4)	Student	0.5248
UN(5,5)	Student	1.6156
UN(6,1)	Student	0.3729
UN(6,2)	Student	0.3554
UN(6,3)	Student	0.1394
UN(6,4)	Student	0.3515
UN(6,5)	Student	0.1965
UN(6,6)	Student	0.8853
UN(7,1)	Student	0.3071
UN(7,2)	Student	0.3479
UN(7,3)	Student	0.02399

UN(7,4)	Student	0.1612
UN(7,5)	Student	0.3254
UN(7,6)	Student	0.3075
UN(7,7)	Student	2.1271
UN(8,1)	Student	0.1918
UN(8,2)	Student	0.08647
UN(8,3)	Student	0.1226
UN(8,4)	Student	0.3873
UN(8,5)	Student	0.09661
UN(8,6)	Student	0.1069
UN(8,7)	Student	0.5222
UN(8,8)	Student	2.2734
UN(9,1)	Student	0.3945
UN(9,2)	Student	0.1547
UN(9,3)	Student	0.2853
UN(9,4)	Student	0.1376
UN(9,5)	Student	0.1896
UN(9,6)	Student	0.5897
UN(9,7)	Student	0.5051
UN(9,8)	Student	0.5793
UN(9,9)	Student	1.5748
UN(10,1)	Student	0.5176
UN(10,2)	Student	0.1200
UN(10,3)	Student	0.08563
UN(10,4)	Student	0.3700
UN(10,5)	Student	0.2303
UN(10,6)	Student	0.4920
UN(10,7)	Student	0.5105
UN(10,8)	Student	0.2990
UN(10,9)	Student	0.5998
UN(10,10)	Student	1.4386

Fit Statistics

-2 Res Log Likelihood	5535.2
AIC (smaller is better)	5645.2
AICC (smaller is better)	5649.0
BIC (smaller is better)	5825.8

Null Model Likelihood Ratio Test

DF	Chi-Square	Pr > ChiSq
54	304.29	<.0001

Solution for Fixed Effects

Effect	Estimate	Standard Error	DF	t Value	Pr > |t|
Intercept	1.3671	0.1824	888	7.50	<.0001
KLnKLOC	0.02557	0.05130	732	0.50	0.6184
DLnDsTim	0.1319	0.04511	932	2.92	0.0036
ELnDRR	-0.06354	0.06572	798	-0.97	0.3339
FLnDDDR	0.1114	0.03790	587	2.94	0.0034
GLnCoTim	0.2263	0.06315	929	3.58	0.0004
HLnCRR	-0.2120	0.06377	791	-3.33	0.0009

ILnDDCR	-0.02155	0.03887	603	-0.55	0.5795
JLnDDCm	0.1183	0.02285	1426	5.18	<.0001

Type 3 Tests of Fixed Effects

Effect	Num DF	Den DF	F Value	Pr > F
KLnKLOC	1	732	0.25	0.6184
DLnDsTim	1	932	8.54	0.0036
ELnDRR	1	798	0.93	0.3339
FLnDDDR	1	587	8.64	0.0034
GLnCoTim	1	929	12.84	0.0004
HLnCRR	1	791	11.06	0.0009
ILnDDCR	1	603	0.31	0.5795
JLnDDCm	1	1426	26.79	<.0001

C.12 COMPILE MIXED MODELS WITH INTERACTIONS FOR (PSPB, C, NOOUTLIERS)

The Mixed Procedure

Dimensions

Covariance Parameters	55
Columns in X	15
Columns in Z	0
Subjects	197
Max Obs Per Subject	10
Observations Used	1705
Observations Not Used	0
Total Observations	1705

Iteration History

Iteration	Evaluations	-2 Res Log Like	Criterion
0	1	5824.64410745	
1	2	5520.98269895	0.00090660
2	1	5519.78039793	0.00002943
3	1	5519.74387433	0.00000006
4	1	5519.74380373	0.00000000

Convergence criteria met.

Estimated R Matrix for Student 1995m10

Row	Col1	Col2	Col3	Col4	Col5	Col6	Col7
1	2.0908	0.5079	0.3893	0.3106	0.3989	0.3625	0.3263
2	0.5079	1.9523	0.3283	0.7708	0.5939	0.3544	0.3942
3	0.3893	0.3283	1.1398	0.2381	0.2313	0.1799	0.04148

4	0.3106	0.7708	0.2381	2.3948	0.5222	0.3485	0.2583
5	0.3989	0.5939	0.2313	0.5222	1.5854	0.1814	0.4180
6	0.3625	0.3544	0.1799	0.3485	0.1814	0.8546	0.2515
7	0.3263	0.3942	0.04148	0.2583	0.4180	0.2515	2.1897

Estimated R Correlation Matrix for Student 1995m10

Row	Col1	Col2	Col3	Col4	Col5	Col6	Col7
1	1.0000	0.2514	0.2522	0.1388	0.2191	0.2712	0.1525
2	0.2514	1.0000	0.2201	0.3565	0.3376	0.2744	0.1907
3	0.2522	0.2201	1.0000	0.1441	0.1721	0.1822	0.02626
4	0.1388	0.3565	0.1441	1.0000	0.2680	0.2436	0.1128
5	0.2191	0.3376	0.1721	0.2680	1.0000	0.1558	0.2244
6	0.2712	0.2744	0.1822	0.2436	0.1558	1.0000	0.1838
7	0.1525	0.1907	0.02626	0.1128	0.2244	0.1838	1.0000

Covariance Parameter Estimates

Cov Parm	Subject	Estimate
UN(1,1)	Student	2.0908
UN(2,1)	Student	0.5079
UN(2,2)	Student	1.9523
UN(3,1)	Student	0.3893
UN(3,2)	Student	0.3283
UN(3,3)	Student	1.1398
UN(4,1)	Student	0.3106
UN(4,2)	Student	0.7708
UN(4,3)	Student	0.2381
UN(4,4)	Student	2.3948
UN(5,1)	Student	0.3989
UN(5,2)	Student	0.5939
UN(5,3)	Student	0.2313
UN(5,4)	Student	0.5222
UN(5,5)	Student	1.5854
UN(6,1)	Student	0.3625
UN(6,2)	Student	0.3544
UN(6,3)	Student	0.1799
UN(6,4)	Student	0.3485
UN(6,5)	Student	0.1814
UN(6,6)	Student	0.8546
UN(7,1)	Student	0.3263
UN(7,2)	Student	0.3942
UN(7,3)	Student	0.04148
UN(7,4)	Student	0.2583
UN(7,5)	Student	0.4180
UN(7,6)	Student	0.2515
UN(7,7)	Student	2.1897
UN(8,1)	Student	0.2523
UN(8,2)	Student	0.07714
UN(8,3)	Student	0.09264
UN(8,4)	Student	0.4446
UN(8,5)	Student	0.1111
UN(8,6)	Student	0.1031
UN(8,7)	Student	0.3983
UN(8,8)	Student	2.2943

UN(9,1)	Student	0.3639
UN(9,2)	Student	0.1036
UN(9,3)	Student	0.3271
UN(9,4)	Student	0.07943
UN(9,5)	Student	0.2018
UN(9,6)	Student	0.5197
UN(9,7)	Student	0.4038
UN(9,8)	Student	0.5611
UN(9,9)	Student	1.5201
UN(10,1)	Student	0.4778
UN(10,2)	Student	0.1056
UN(10,3)	Student	0.07142
UN(10,4)	Student	0.3319
UN(10,5)	Student	0.2101
UN(10,6)	Student	0.4858
UN(10,7)	Student	0.4821
UN(10,8)	Student	0.2997
UN(10,9)	Student	0.5543
UN(10,10)	Student	1.3860

Fit Statistics

-2 Res Log Likelihood	5519.7
AIC (smaller is better)	5629.7
AICC (smaller is better)	5633.5
BIC (smaller is better)	5810.3

Null Model Likelihood Ratio Test

DF	Chi-Square	Pr > ChiSq
54	304.90	<.0001

Solution for Fixed Effects

Effect	Estimate	Standard Error	DF	t Value	Pr > |t|
Intercept	1.3656	0.4502	922	3.03	0.0025
KLnKLOC	-0.2079	0.1521	952	-1.37	0.1720
DLnDsTim	0.03645	0.1429	821	0.26	0.7987
ELnDRR	-0.3908	0.1327	575	-2.94	0.0034
FLnDDDR	-0.1881	0.1315	440	-1.43	0.1535
GLnCoTim	-0.01706	0.1251	1070	-0.14	0.8916
HLnCRR	-0.2283	0.06562	766	-3.48	0.0005
ILnDDCR	0.4750	0.1275	530	3.72	0.0002
JLnDDCm	-0.07373	0.08686	1231	-0.85	0.3961
KLnKLOC*DLnDsTim	-0.03506	0.05324	928	-0.66	0.5104
KLnKLOC*ELnDRR	-0.1337	0.04638	612	-2.88	0.0041
KLnKLOC*FLnDDDR	-0.1318	0.05425	513	-2.43	0.0155
KLnKLOC*ILnDDCR	0.2199	0.05230	712	4.20	<.0001
GLnCoTim*JLnDDCm	0.07396	0.03358	1167	2.20	0.0278
KLnK*GLnC*HLnC*ILnDD	-0.00818	0.003855	657	-2.12	0.0342

Type 3 Tests of Fixed Effects

Num	Den

Effect	DF	DF	F Value	Pr > F
KLnKLOC	1	952	1.87	0.1720
DLnDsTim	1	821	0.07	0.7987
ELnDRR	1	575	8.67	0.0034
FLnDDDR	1	440	2.04	0.1535
GLnCoTim	1	1070	0.02	0.8916
HLnCRR	1	766	12.10	0.0005
ILnDDCR	1	530	13.88	0.0002
JLnDDCm	1	1231	0.72	0.3961
KLnKLOC*DLnDsTim	1	928	0.43	0.5104
KLnKLOC*ELnDRR	1	612	8.31	0.0041
KLnKLOC*FLnDDDR	1	513	5.90	0.0155
KLnKLOC*ILnDDCR	1	712	17.68	<.0001
GLnCoTim*JLnDDCm	1	1167	4.85	0.0278
KLnK*GLnC*HLnC*ILnDD	1	657	4.50	0.0342

C.13 COMPILE MIXED MODELS FOR (PSPB, C++, OUTLIERS)

The Mixed Procedure

Dimensions

Covariance Parameters	55
Columns in X	9
Columns in Z	0
Subjects	108
Max Obs Per Subject	10
Observations Used	920
Observations Not Used	0
Total Observations	920

Iteration History

Iteration	Evaluations	-2 Res Log Like	Criterion
0	1	3088.72979503	
1	2	2967.24021713	0.00017340
2	1	2967.12359823	0.00000080
3	1	2967.12308002	0.00000000

Convergence criteria met.

Estimated R Matrix for Student 1995p6

Row	Col1	Col2	Col3	Col4	Col5	Col6	Col7	Col8	Col9
1	1.5747	0.07598	0.3105	0.2739	0.08546	0.1535	0.3859	0.2643	0.1871
2	0.07598	1.7492	0.6261	0.2981	0.4030	0.5539	0.2012	0.1561	0.3617
3	0.3105	0.6261	1.5053	-0.06126	0.4040	0.3995	0.04650	0.2590	0.2753
4	0.2739	0.2981	-0.06126	2.0031	-0.09393	0.1597	0.3243	0.002969	0.1348

5	0.08546	0.4030	0.4040	-0.09393	1.7179	0.4640	0.3274	0.2291	0.1158

Estimated R
Matrix for
Student 1995p6

Row	Col10
1	0.04361
2	0.08112
3	0.1510
4	-0.1360
5	0.4102

Estimated R Matrix for Student 1995p6

Row	Col1	Col2	Col3	Col4	Col5	Col6	Col7	Col8	Col9
6	0.1535	0.5539	0.3995	0.1597	0.4640	1.0476	0.02608	0.3152	0.2119
7	0.3859	0.2012	0.04650	0.3243	0.3274	0.02608	2.1947	0.1963	0.4105
8	0.2643	0.1561	0.2590	0.002969	0.2291	0.3152	0.1963	2.0493	0.09875
9	0.1871	0.3617	0.2753	0.1348	0.1158	0.2119	0.4105	0.09875	1.2246
10	0.04361	0.08112	0.1510	-0.1360	0.4102	0.1668	0.2386	0.1423	0.07995

Estimated R
Matrix for
Student 1995p6

Row	Col10
6	0.1668
7	0.2386
8	0.1423
9	0.07995
10	1.0990

Estimated R Correlation Matrix for Student 1995p6

Row	Col1	Col2	Col3	Col4	Col5	Col6	Col7	Col8	Col9
1	1.0000	0.04578	0.2017	0.1542	0.05196	0.1195	0.2076	0.1471	0.1347
2	0.04578	1.0000	0.3859	0.1592	0.2325	0.4092	0.1027	0.08243	0.2471
3	0.2017	0.3859	1.0000	-0.03528	0.2512	0.3181	0.02559	0.1475	0.2028
4	0.1542	0.1592	-0.03528	1.0000	-0.05064	0.1103	0.1547	0.001465	0.08606

Estimated R
Correlation
Matrix for
Student 1995p6

Row	Col10
1	0.03315
2	0.05851
3	0.1174
4	-0.09164

Estimated R Correlation Matrix for Student 1995p6

Row	Col1	Col2	Col3	Col4	Col5	Col6	Col7	Col8	Col9
5	0.05196	0.2325	0.2512	-0.05064	1.0000	0.3459	0.1686	0.1221	0.07982
6	0.1195	0.4092	0.3181	0.1103	0.3459	1.0000	0.01720	0.2151	0.1871
7	0.2076	0.1027	0.02559	0.1547	0.1686	0.01720	1.0000	0.09254	0.2504
8	0.1471	0.08243	0.1475	0.001465	0.1221	0.2151	0.09254	1.0000	0.06234
9	0.1347	0.2471	0.2028	0.08606	0.07982	0.1871	0.2504	0.06234	1.0000
10	0.03315	0.05851	0.1174	-0.09164	0.2985	0.1554	0.1536	0.09485	0.06892

Estimated R Correlation Matrix for Student 1995p6

Row	Col10
5	0.2985
6	0.1554
7	0.1536
8	0.09485
9	0.06892
10	1.0000

Covariance Parameter Estimates

Cov Parm	Subject	Estimate
UN(1,1)	Student	1.5747
UN(2,1)	Student	0.07598
UN(2,2)	Student	1.7492
UN(3,1)	Student	0.3105
UN(3,2)	Student	0.6261
UN(3,3)	Student	1.5053
UN(4,1)	Student	0.2739
UN(4,2)	Student	0.2981
UN(4,3)	Student	-0.06126
UN(4,4)	Student	2.0031
UN(5,1)	Student	0.08546
UN(5,2)	Student	0.4030
UN(5,3)	Student	0.4040
UN(5,4)	Student	-0.09393
UN(5,5)	Student	1.7179
UN(6,1)	Student	0.1535
UN(6,2)	Student	0.5539
UN(6,3)	Student	0.3995
UN(6,4)	Student	0.1597
UN(6,5)	Student	0.4640
UN(6,6)	Student	1.0476
UN(7,1)	Student	0.3859
UN(7,2)	Student	0.2012
UN(7,3)	Student	0.04650
UN(7,4)	Student	0.3243
UN(7,5)	Student	0.3274
UN(7,6)	Student	0.02608
UN(7,7)	Student	2.1947

UN(8,1)	Student	0.2643
UN(8,2)	Student	0.1561
UN(8,3)	Student	0.2590
UN(8,4)	Student	0.002969
UN(8,5)	Student	0.2291
UN(8,6)	Student	0.3152
UN(8,7)	Student	0.1963
UN(8,8)	Student	2.0493
UN(9,1)	Student	0.1871
UN(9,2)	Student	0.3617
UN(9,3)	Student	0.2753
UN(9,4)	Student	0.1348
UN(9,5)	Student	0.1158
UN(9,6)	Student	0.2119
UN(9,7)	Student	0.4105
UN(9,8)	Student	0.09875
UN(9,9)	Student	1.2246
UN(10,1)	Student	0.04361
UN(10,2)	Student	0.08112
UN(10,3)	Student	0.1510
UN(10,4)	Student	-0.1360
UN(10,5)	Student	0.4102
UN(10,6)	Student	0.1668
UN(10,7)	Student	0.2386
UN(10,8)	Student	0.1423
UN(10,9)	Student	0.07995
UN(10,10)	Student	1.0990

Fit Statistics

-2 Res Log Likelihood	2967.1
AIC (smaller is better)	3077.1
AICC (smaller is better)	3084.3
BIC (smaller is better)	3224.6

Null Model Likelihood Ratio Test

DF	Chi-Square	Pr > ChiSq
54	121.61	<.0001

Solution for Fixed Effects

Effect	Estimate	Standard Error	DF	t Value	Pr > |t|
Intercept	0.4853	0.2296	403	2.11	0.0352
KLnKLOC	-0.09935	0.06654	314	-1.49	0.1364
DLnDsTim	0.04044	0.06038	580	0.67	0.5032
ELnDRR	-0.08726	0.09045	332	-0.96	0.3353
FLnDDDR	0.06032	0.05264	285	1.15	0.2528
GLnCoTim	0.4140	0.08495	605	4.87	<.0001
HLnCRR	-0.1913	0.09277	345	-2.06	0.0400
ILnDDCR	0.1080	0.05886	313	1.84	0.0674
JLnDDCm	0.1319	0.03264	625	4.04	<.0001

Type 3 Tests of Fixed Effects

Effect	Num DF	Den DF	F Value	Pr > F
KLnKLOC	1	314	2.23	0.1364
DLnDsTim	1	580	0.45	0.5032
ELnDRR	1	332	0.93	0.3353
FLnDDDR	1	285	1.31	0.2528
GLnCoTim	1	605	23.75	<.0001
HLnCRR	1	345	4.25	0.0400
ILnDDCR	1	313	3.37	0.0674
JLnDDCm	1	625	16.33	<.0001

C.14 COMPILE MIXED MODELS WITH INTERACTIONS FOR (PSPB, C++, OUTLIERS)

The Mixed Procedure

Dimensions

Covariance Parameters	55
Columns in X	15
Columns in Z	0
Subjects	108
Max Obs Per Subject	10
Observations Used	920
Observations Not Used	0
Total Observations	920

Iteration History

Iteration	Evaluations	-2 Res Log Like	Criterion
0	1	3094.57558895	
1	2	2974.89205832	0.00037643
2	1	2974.63052572	0.00000397
3	1	2974.62789751	0.00000000

Convergence criteria met.

Estimated R Matrix for Student 1995p6

Row	Col1	Col2	Col3	Col4	Col5	Col6	Col7	Col8	Col9
1	1.5770	0.06633	0.2795	0.3032	0.1208	0.1758	0.2972	0.2673	0.1548
2	0.06633	1.6828	0.6151	0.2837	0.4051	0.5476	0.2175	0.2546	0.3747
3	0.2795	0.6151	1.4848	-0.1078	0.3818	0.3759	0.06945	0.3235	0.2712
4	0.3032	0.2837	-0.1078	2.0254	-0.06966	0.1705	0.3905	0.06101	0.1377
5	0.1208	0.4051	0.3818	-0.06966	1.7821	0.5010	0.4324	0.2301	0.1385

Estimated R
Matrix for
Student 1995p6

Row	Col10
1	0.01377
2	0.1294
3	0.1351
4	-0.07869
5	0.4109

Estimated R Matrix for Student 1995p6

Row	Col1	Col2	Col3	Col4	Col5	Col6	Col7	Col8	Col9
6	0.1758	0.5476	0.3759	0.1705	0.5010	1.0486	0.01167	0.3560	0.1600
7	0.2972	0.2175	0.06945	0.3905	0.4324	0.01167	1.9675	0.08781	0.4342
8	0.2673	0.2546	0.3235	0.06101	0.2301	0.3560	0.08781	2.0629	0.2146
9	0.1548	0.3747	0.2712	0.1377	0.1385	0.1600	0.4342	0.2146	1.2343
10	0.01377	0.1294	0.1351	-0.07869	0.4109	0.1554	0.2309	0.1153	0.1434

Estimated R
Matrix for
Student 1995p6

Row	Col10
6	0.1554
7	0.2309
8	0.1153
9	0.1434
10	1.0853

Estimated R Correlation Matrix for Student 1995p6

Row	Col1	Col2	Col3	Col4	Col5	Col6	Col7	Col8	Col9
1	1.0000	0.04072	0.1826	0.1696	0.07206	0.1367	0.1687	0.1482	0.1109
2	0.04072	1.0000	0.3891	0.1537	0.2340	0.4122	0.1195	0.1366	0.2600
3	0.1826	0.3891	1.0000	-0.06219	0.2347	0.3012	0.04064	0.1848	0.2003
4	0.1696	0.1537	-0.06219	1.0000	-0.03667	0.1170	0.1956	0.02985	0.08712

Estimated R
Correlation
Matrix for
Student 1995p6

Row	Col10
1	0.01053
2	0.09573
3	0.1065
4	-0.05307

Estimated R Correlation Matrix for Student 1995p6

Row	Col1	Col2	Col3	Col4	Col5	Col6	Col7	Col8	Col9
5	0.07206	0.2340	0.2347	-0.03667	1.0000	0.3665	0.2309	0.1200	0.09340
6	0.1367	0.4122	0.3012	0.1170	0.3665	1.0000	0.008124	0.2421	0.1406
7	0.1687	0.1195	0.04064	0.1956	0.2309	0.008124	1.0000	0.04359	0.2786
8	0.1482	0.1366	0.1848	0.02985	0.1200	0.2421	0.04359	1.0000	0.1345
9	0.1109	0.2600	0.2003	0.08712	0.09340	0.1406	0.2786	0.1345	1.0000
10	0.01053	0.09573	0.1065	-0.05307	0.2955	0.1457	0.1580	0.07708	0.1239

Estimated R Correlation Matrix for Student 1995p6

Row	Col10
5	0.2955
6	0.1457
7	0.1580
8	0.07708
9	0.1239
10	1.0000

Covariance Parameter Estimates

Cov Parm	Subject	Estimate
UN(1,1)	Student	1.5770
UN(2,1)	Student	0.06633
UN(2,2)	Student	1.6828
UN(3,1)	Student	0.2795
UN(3,2)	Student	0.6151
UN(3,3)	Student	1.4848
UN(4,1)	Student	0.3032
UN(4,2)	Student	0.2837
UN(4,3)	Student	-0.1078
UN(4,4)	Student	2.0254
UN(5,1)	Student	0.1208
UN(5,2)	Student	0.4051
UN(5,3)	Student	0.3818
UN(5,4)	Student	-0.06966
UN(5,5)	Student	1.7821
UN(6,1)	Student	0.1758
UN(6,2)	Student	0.5476
UN(6,3)	Student	0.3759
UN(6,4)	Student	0.1705
UN(6,5)	Student	0.5010
UN(6,6)	Student	1.0486
UN(7,1)	Student	0.2972
UN(7,2)	Student	0.2175
UN(7,3)	Student	0.06945
UN(7,4)	Student	0.3905
UN(7,5)	Student	0.4324
UN(7,6)	Student	0.01167
UN(7,7)	Student	1.9675
UN(8,1)	Student	0.2673

UN(8,2)	Student	0.2546
UN(8,3)	Student	0.3235
UN(8,4)	Student	0.06101
UN(8,5)	Student	0.2301
UN(8,6)	Student	0.3560
UN(8,7)	Student	0.08781
UN(8,8)	Student	2.0629
UN(9,1)	Student	0.1548
UN(9,2)	Student	0.3747
UN(9,3)	Student	0.2712
UN(9,4)	Student	0.1377
UN(9,5)	Student	0.1385
UN(9,6)	Student	0.1600
UN(9,7)	Student	0.4342
UN(9,8)	Student	0.2146
UN(9,9)	Student	1.2343
UN(10,1)	Student	0.01377
UN(10,2)	Student	0.1294
UN(10,3)	Student	0.1351
UN(10,4)	Student	-0.07869
UN(10,5)	Student	0.4109
UN(10,6)	Student	0.1554
UN(10,7)	Student	0.2309
UN(10,8)	Student	0.1153
UN(10,9)	Student	0.1434
UN(10,10)	Student	1.0853

Fit Statistics

-2 Res Log Likelihood	2974.6
AIC (smaller is better)	3084.6
AICC (smaller is better)	3091.9
BIC (smaller is better)	3232.1

Null Model Likelihood Ratio Test

DF	Chi-Square	Pr > ChiSq
54	119.95	<.0001

Solution for Fixed Effects

Effect	Estimate	Standard Error	DF	t Value	Pr > |t|
Intercept	-0.2592	0.5276	420	-0.49	0.6234
KLnKLOC	-0.3089	0.1946	526	-1.59	0.1130
DLnDsTim	0.2792	0.1656	601	1.69	0.0924
ELnDRR	-0.07527	0.1605	272	-0.47	0.6395
FLnDDDR	0.1593	0.1656	246	0.96	0.3369
GLnCoTim	0.5575	0.1679	429	3.32	0.0010
HLnCRR	-0.2529	0.09494	371	-2.66	0.0081
ILnDDCR	0.1633	0.1703	288	0.96	0.3383
JLnDDCm	0.2708	0.1130	495	2.40	0.0170
KLnKLOC*DLnDsTim	0.09675	0.06331	624	1.53	0.1269
KLnKLOC*ELnDRR	-0.00980	0.06428	310	-0.15	0.8789
KLnKLOC*FLnDDDR	0.06365	0.07008	300	0.91	0.3644

KLnKLOC*ILnDDCR	0.08133	0.07547	320	1.08	0.2820
GLnCoTim*JLnDDCm	-0.06841	0.04483	546	-1.53	0.1276
KLnK*GLnC*HLnC*ILnDD	-0.01646	0.003980	335	-4.13	<.0001

Type 3 Tests of Fixed Effects

Effect	Num DF	Den DF	F Value	Pr > F
KLnKLOC	1	526	2.52	0.1130
DLnDsTim	1	601	2.84	0.0924
ELnDRR	1	272	0.22	0.6395
FLnDDDR	1	246	0.93	0.3369
GLnCoTim	1	429	11.03	0.0010
HLnCRR	1	371	7.10	0.0081
ILnDDCR	1	288	0.92	0.3383
JLnDDCm	1	495	5.74	0.0170
KLnKLOC*DLnDsTim	1	624	2.34	0.1269
KLnKLOC*ELnDRR	1	310	0.02	0.8789
KLnKLOC*FLnDDDR	1	300	0.83	0.3644
KLnKLOC*ILnDDCR	1	320	1.16	0.2820
GLnCoTim*JLnDDCm	1	546	2.33	0.1276
KLnK*GLnC*HLnC*ILnDD	1	335	17.09	<.0001

C.15 COMPILE MIXED MODELS FOR (PSPB, C++, NOOUTLIERS)

The Mixed Procedure

Dimensions

Covariance Parameters	55
Columns in X	9
Columns in Z	0
Subjects	108
Max Obs Per Subject	10
Observations Used	892
Observations Not Used	0
Total Observations	892

Iteration History

Iteration	Evaluations	-2 Res Log Like	Criterion
0	1	2957.10078571	
1	2	2850.62169063	0.00042974
2	1	2850.34384472	0.00000340
3	1	2850.34174635	0.00000000

Convergence criteria met.

Estimated R Matrix for Student 1995p6

Row	Col1	Col2	Col3	Col4	Col5	Col6	Col7	Col8	Col9
1	1.5403	0.06337	0.2276	0.2917	0.09685	0.1518	0.2307	0.1566	0.01517
2	0.06337	1.6605	0.5676	0.2597	0.3472	0.4645	0.08267	0.3040	0.1366
3	0.2276	0.5676	1.4513	-0.1147	0.4336	0.3466	0.2081	0.1918	0.1494
4	0.2917	0.2597	-0.1147	1.9647	-0.07842	0.2344	-0.02442	0.1089	-0.1125
5	0.09685	0.3472	0.4336	-0.07842	1.6416	0.4282	0.1400	0.03356	0.2984
6	0.1518	0.4645	0.3466	0.2344	0.4282	1.0390	0.2820	0.1838	0.1637
7	0.2307	0.08267	0.2081	-0.02442	0.1400	0.2820	1.9705	0.05804	0.1256
8	0.1566	0.3040	0.1918	0.1089	0.03356	0.1838	0.05804	1.2002	0.08885
9	0.01517	0.1366	0.1494	-0.1125	0.2984	0.1637	0.1256	0.08885	1.1200

Estimated R Correlation Matrix for Student 1995p6

Row	Col1	Col2	Col3	Col4	Col5	Col6	Col7	Col8	Col9
1	1.0000	0.03962	0.1522	0.1677	0.06091	0.1200	0.1324	0.1152	0.01155
2	0.03962	1.0000	0.3657	0.1438	0.2103	0.3536	0.04570	0.2153	0.1002
3	0.1522	0.3657	1.0000	-0.06794	0.2809	0.2823	0.1231	0.1454	0.1172
4	0.1677	0.1438	-0.06794	1.0000	-0.04367	0.1641	-0.01241	0.07092	-0.07584
5	0.06091	0.2103	0.2809	-0.04367	1.0000	0.3279	0.07786	0.02391	0.2200

Estimated R Correlation Matrix for Student 1995p6

Row	Col1	Col2	Col3	Col4	Col5	Col6	Col7	Col8	Col9
6	0.1200	0.3536	0.2823	0.1641	0.3279	1.0000	0.1971	0.1646	0.1517
7	0.1324	0.04570	0.1231	-0.01241	0.07786	0.1971	1.0000	0.03774	0.08453
8	0.1152	0.2153	0.1454	0.07092	0.02391	0.1646	0.03774	1.0000	0.07663
9	0.01155	0.1002	0.1172	-0.07584	0.2200	0.1517	0.08453	0.07663	1.0000

Covariance Parameter Estimates

Cov Parm	Subject	Estimate
UN(1,1)	Student	1.5403
UN(2,1)	Student	0.06337
UN(2,2)	Student	1.6605
UN(3,1)	Student	0.2276
UN(3,2)	Student	0.5676
UN(3,3)	Student	1.4513
UN(4,1)	Student	0.2917
UN(4,2)	Student	0.2597
UN(4,3)	Student	-0.1147
UN(4,4)	Student	1.9647
UN(5,1)	Student	0.09685
UN(5,2)	Student	0.3472
UN(5,3)	Student	0.4336
UN(5,4)	Student	-0.07842
UN(5,5)	Student	1.6416
UN(6,1)	Student	0.1518
UN(6,2)	Student	0.4645
UN(6,3)	Student	0.3466
UN(6,4)	Student	0.2344
UN(6,5)	Student	0.4282
UN(6,6)	Student	1.0390
UN(7,1)	Student	0.3080

UN(7,2)	Student	0.1098
UN(7,3)	Student	-0.03507
UN(7,4)	Student	0.4120
UN(7,5)	Student	0.5200
UN(7,6)	Student	-0.03202
UN(7,7)	Student	1.9672
UN(8,1)	Student	0.2307
UN(8,2)	Student	0.08267
UN(8,3)	Student	0.2081
UN(8,4)	Student	-0.02442
UN(8,5)	Student	0.1400
UN(8,6)	Student	0.2820
UN(8,7)	Student	0.02473
UN(8,8)	Student	1.9705
UN(9,1)	Student	0.1566
UN(9,2)	Student	0.3040
UN(9,3)	Student	0.1918
UN(9,4)	Student	0.1089
UN(9,5)	Student	0.03356
UN(9,6)	Student	0.1838
UN(9,7)	Student	0.3398
UN(9,8)	Student	0.05804
UN(9,9)	Student	1.2002
UN(10,1)	Student	0.01517
UN(10,2)	Student	0.1366
UN(10,3)	Student	0.1494
UN(10,4)	Student	-0.1125
UN(10,5)	Student	0.2984
UN(10,6)	Student	0.1637
UN(10,7)	Student	0.2443
UN(10,8)	Student	0.1256
UN(10,9)	Student	0.08885
UN(10,10)	Student	1.1200

Fit Statistics

-2 Res Log Likelihood	2850.3
AIC (smaller is better)	2960.3
AICC (smaller is better)	2967.8
BIC (smaller is better)	3107.9

Null Model Likelihood Ratio Test

DF	Chi-Square	Pr > ChiSq
54	106.76	<.0001

Solution for Fixed Effects

Effect	Estimate	Standard Error	DF	t Value	Pr > |t|
Intercept	0.7437	0.2335	395	3.18	0.0016
KLnKLOC	0.006278	0.06841	315	0.09	0.9269
DLnDsTim	0.04883	0.06202	588	0.79	0.4314
ELnDRR	-0.1195	0.09017	330	-1.32	0.1862
FLnDDDR	0.08317	0.05242	283	1.59	0.1137

GLnCoTim	0.3614	0.08588	577	4.21	<.0001
HLnCRR	-0.1916	0.09210	348	-2.08	0.0383
ILnDDCR	0.1057	0.05840	309	1.81	0.0713
JLnDDCm	0.1336	0.03310	610	4.04	<.0001

Type 3 Tests of Fixed Effects

Effect	Num DF	Den DF	F Value	Pr > F
KLnKLOC	1	315	0.01	0.9269
DLnDsTim	1	588	0.62	0.4314
ELnDRR	1	330	1.76	0.1862
FLnDDDR	1	283	2.52	0.1137
GLnCoTim	1	577	17.71	<.0001
HLnCRR	1	348	4.33	0.0383
ILnDDCR	1	309	3.28	0.0713
JLnDDCm	1	610	16.29	<.0001

C.16 COMPILE MIXED MODELS WITH INTERACTIONS FOR (PSPB, C++, NOUTLIERS)

The Mixed Procedure

Dimensions

Covariance Parameters	55
Columns in X	15
Columns in Z	0
Subjects	108
Max Obs Per Subject	10
Observations Used	884
Observations Not Used	0
Total Observations	884

Iteration History

Iteration	Evaluations	-2 Res Log Like	Criterion
0	1	2940.84127538	
1	2	2827.01976447	0.00082158
2	1	2826.47960430	0.00001167
3	1	2826.47233962	0.00000000

Convergence criteria met.

Estimated R Matrix for Student 1995p6

Row	Col1	Col2	Col3	Col4	Col5	Col6	Col7	Col8
1	1.5646	0.1025	0.2871	0.1027	0.1633	0.2217	0.1248	0.006519
2	0.1025	1.6125	0.2463	0.3276	0.4565	0.1948	0.2913	0.2308

3	0.2871	0.2463	1.9349	-0.07419	0.2174	-0.06249	0.06371	0.07906
4	0.1027	0.3276	-0.07419	1.7292	0.4215	0.1177	0.01137	0.3638
5	0.1633	0.4565	0.2174	0.4215	0.9927	0.3383	0.1209	0.1191
6	0.2217	0.1948	-0.06249	0.1177	0.3383	1.9334	0.1725	0.08857
7	0.1248	0.2913	0.06371	0.01137	0.1209	0.1725	1.2023	0.1139
8	0.006519	0.2308	0.07906	0.3638	0.1191	0.08857	0.1139	0.9706

Estimated R Correlation Matrix for Student 1995p6

Row	Col1	Col2	Col3	Col4	Col5	Col6	Col7	Col8
1	1.0000	0.06450	0.1650	0.06244	0.1311	0.1275	0.09097	0.005290
2	0.06450	1.0000	0.1394	0.1962	0.3608	0.1103	0.2092	0.1845
3	0.1650	0.1394	1.0000	-0.04056	0.1569	-0.03231	0.04177	0.05769
4	0.06244	0.1962	-0.04056	1.0000	0.3217	0.06438	0.007886	0.2808
5	0.1311	0.3608	0.1569	0.3217	1.0000	0.2442	0.1106	0.1213
6	0.1275	0.1103	-0.03231	0.06438	0.2442	1.0000	0.1131	0.06466
7	0.09097	0.2092	0.04177	0.007886	0.1106	0.1131	1.0000	0.1055
8	0.005290	0.1845	0.05769	0.2808	0.1213	0.06466	0.1055	1.0000

Covariance Parameter Estimates

Cov Parm	Subject	Estimate
UN(1,1)	Student	1.5646
UN(2,1)	Student	0.1025
UN(2,2)	Student	1.6125
UN(3,1)	Student	0.3550
UN(3,2)	Student	0.7686
UN(3,3)	Student	1.5318
UN(4,1)	Student	0.2871
UN(4,2)	Student	0.2463
UN(4,3)	Student	-0.04275
UN(4,4)	Student	1.9349
UN(5,1)	Student	0.1027
UN(5,2)	Student	0.3276
UN(5,3)	Student	0.4282
UN(5,4)	Student	-0.07419
UN(5,5)	Student	1.7292
UN(6,1)	Student	0.1633
UN(6,2)	Student	0.4565
UN(6,3)	Student	0.4171
UN(6,4)	Student	0.2174
UN(6,5)	Student	0.4215
UN(6,6)	Student	0.9927
UN(7,1)	Student	0.2577
UN(7,2)	Student	0.1078
UN(7,3)	Student	0.04398
UN(7,4)	Student	0.3607
UN(7,5)	Student	0.4866
UN(7,6)	Student	-0.04656
UN(7,7)	Student	1.9244
UN(8,1)	Student	0.2217
UN(8,2)	Student	0.1948
UN(8,3)	Student	0.3189
UN(8,4)	Student	-0.06249
UN(8,5)	Student	0.1177

UN(8,6)	Student	0.3383
UN(8,7)	Student	-0.07570
UN(8,8)	Student	1.9334
UN(9,1)	Student	0.1248
UN(9,2)	Student	0.2913
UN(9,3)	Student	0.3054
UN(9,4)	Student	0.06371
UN(9,5)	Student	0.01137
UN(9,6)	Student	0.1209
UN(9,7)	Student	0.2926
UN(9,8)	Student	0.1725
UN(9,9)	Student	1.2023
UN(10,1)	Student	0.006519
UN(10,2)	Student	0.2308
UN(10,3)	Student	0.2763
UN(10,4)	Student	0.07906
UN(10,5)	Student	0.3638
UN(10,6)	Student	0.1191
UN(10,7)	Student	0.2268
UN(10,8)	Student	0.08857
UN(10,9)	Student	0.1139
UN(10,10)	Student	0.9706

Fit Statistics

-2 Res Log Likelihood	2826.5
AIC (smaller is better)	2936.5
AICC (smaller is better)	2944.0
BIC (smaller is better)	3084.0

Null Model Likelihood Ratio Test

DF	Chi-Square	Pr > ChiSq
54	114.37	<.0001

Solution for Fixed Effects

Effect	Estimate	Standard Error	DF	t Value	Pr > |t|
Intercept	-0.4214	0.5504	377	-0.77	0.4444
KLnKLOC	-0.1755	0.2082	461	-0.84	0.3997
DLnDsTim	0.4791	0.1735	543	2.76	0.0059
ELnDRR	0.08923	0.1654	239	0.54	0.5901
FLnDDDR	-0.06682	0.1813	220	-0.37	0.7128
GLnCoTim	0.6692	0.1707	365	3.92	0.0001
HLnCRR	-0.2486	0.09435	354	-2.63	0.0088
ILnDDCR	0.07726	0.1790	257	0.43	0.6664
JLnDDCm	0.4127	0.1163	438	3.55	0.0004
KLnKLOC*DLnDsTim	0.1741	0.06804	549	2.56	0.0108
KLnKLOC*ELnDRR	0.07545	0.06942	282	1.09	0.2780
KLnKLOC*FLnDDDR	-0.04014	0.07889	279	-0.51	0.6113
KLnKLOC*ILnDDCR	0.01309	0.08237	313	0.16	0.8738
GLnCoTim*JLnDDCm	-0.1208	0.04697	487	-2.57	0.0104
KLnK*GLnC*HLnC*ILnDD	-0.01073	0.004948	282	-2.17	0.0310

Type 3 Tests of Fixed Effects

Effect	Num DF	Den DF	F Value	Pr > F
KLnKLOC	1	461	0.71	0.3997
DLnDsTim	1	543	7.63	0.0059
ELnDRR	1	239	0.29	0.5901
FLnDDDR	1	220	0.14	0.7128
GLnCoTim	1	365	15.37	0.0001
HLnCRR	1	354	6.94	0.0088
ILnDDCR	1	257	0.19	0.6664
JLnDDCm	1	438	12.60	0.0004
KLnKLOC*DLnDsTim	1	549	6.54	0.0108
KLnKLOC*ELnDRR	1	282	1.18	0.2780
KLnKLOC*FLnDDDR	1	279	0.26	0.6113
KLnKLOC*ILnDDCR	1	313	0.03	0.8738
GLnCoTim*JLnDDCm	1	487	6.62	0.0104
KLnK*GLnC*HLnC*ILnDD	1	282	4.70	0.0310

C.17 RANDOM COEFFICIENT MIXED MODELS BY ASSIGNMENT FOR (PSPB, C, ALLTEN)

The Mixed Procedure

Solution for Fixed Effects

Effect	Estimate	Standard Error	DF	t Value	Pr > |t|
Intercept	3.6587	0.1060	109	34.51	<.0001
Assignment	-0.1749	0.01373	109	-12.74	<.0001

Solution for Random Effects

Effect	Student	Estimate	Std Err Pred	DF	t Value	Pr > |t|
Intercept	1996h12	-0.3048	0.4466	16.8	-0.68	0.5042
Assignment	1996h12	0.007978	0.04891	2.36	0.16	0.8833
Intercept	1996h3	-0.3748	0.4466	16.8	-0.84	0.4131
Assignment	1996h3	-0.00108	0.04891	2.36	-0.02	0.9842
Intercept	1996i3	0.1211	0.4466	16.8	0.27	0.7895
Assignment	1996i3	-0.01903	0.04891	2.36	-0.39	0.7295
Intercept	1996i4	-0.05369	0.4466	16.8	-0.12	0.9057
Assignment	1996i4	-0.01511	0.04891	2.36	-0.31	0.7825
Intercept	1997c10	-0.8873	0.4466	16.8	-1.99	0.0636
Assignment	1997c10	0.02629	0.04891	2.36	0.54	0.6373
Intercept	1997d1	0.1048	0.4466	16.8	0.23	0.8172
Assignment	1997d1	0.003319	0.04891	2.36	0.07	0.9512
Intercept	1997e1	-0.4253	0.4466	16.8	-0.95	0.3545
Assignment	1997e1	0.005557	0.04891	2.36	0.11	0.9185
Intercept	1997e12	0.2007	0.4466	16.8	0.45	0.6588

Assignment	1997e12	0.005833	0.04891	2.36	0.12	0.9145
Intercept	1997e13	1.0741	0.4466	16.8	2.41	0.0280
Assignment	1997e13	-0.03342	0.04891	2.36	-0.68	0.5553
Intercept	1997g14	-0.5965	0.4466	16.8	-1.34	0.1995
Assignment	1997g14	0.02908	0.04891	2.36	0.59	0.6041
Intercept	1997i17	0.2230	0.4466	16.8	0.50	0.6241
Assignment	1997i17	-0.00437	0.04891	2.36	-0.09	0.9359
Intercept	1997i19	-0.1334	0.4466	16.8	-0.30	0.7688
Assignment	1997i19	-0.03205	0.04891	2.36	-0.66	0.5704
Intercept	1997i2	0.5943	0.4466	16.8	1.33	0.2011
Assignment	1997i2	0.001104	0.04891	2.36	0.02	0.9838
Intercept	1997i5	1.2558	0.4466	16.8	2.81	0.0121
Assignment	1997i5	-0.02423	0.04891	2.36	-0.50	0.6626
Intercept	1997i8	0.05956	0.4466	16.8	0.13	0.8955
Assignment	1997i8	0.006022	0.04891	2.36	0.12	0.9117
Intercept	1998f2	0.05922	0.4466	16.8	0.13	0.8961
Assignment	1998f2	-0.00478	0.04891	2.36	-0.10	0.9299
Intercept	1998m12	-0.2962	0.4466	16.8	-0.66	0.5161
Assignment	1998m12	-0.00882	0.04891	2.36	-0.18	0.8713
Intercept	1998m14	0.6531	0.4466	16.8	1.46	0.1621
Assignment	1998m14	-0.01162	0.04891	2.36	-0.24	0.8312
Intercept	1998m20	-0.5544	0.4466	16.8	-1.24	0.2316
Assignment	1998m20	-0.01602	0.04891	2.36	-0.33	0.7701
Intercept	1998m4	0.6423	0.4466	16.8	1.44	0.1688
Assignment	1998m4	-0.00457	0.04891	2.36	-0.09	0.9328
Intercept	1998m6	0.08284	0.4466	16.8	0.19	0.8551
Assignment	1998m6	-0.00278	0.04891	2.36	-0.06	0.9591
Intercept	1998w2	-0.5866	0.4466	16.8	-1.31	0.2067
Assignment	1998w2	-0.01397	0.04891	2.36	-0.29	0.7983
Intercept	1998w8	-0.2172	0.4466	16.8	-0.49	0.6330
Assignment	1998w8	0.009547	0.04891	2.36	0.20	0.8608
Intercept	1998x3	0.4032	0.4466	16.8	0.90	0.3794
Assignment	1998x3	-0.00787	0.04891	2.36	-0.16	0.8850
Intercept	1998x4	0.3560	0.4466	16.8	0.80	0.4365
Assignment	1998x4	-0.01458	0.04891	2.36	-0.30	0.7899
Intercept	1998x6	-0.3372	0.4466	16.8	-0.76	0.4607
Assignment	1998x6	-0.00973	0.04891	2.36	-0.20	0.8582
Intercept	1999g1	0.4789	0.4466	16.8	1.07	0.2988
Assignment	1999g1	-0.01296	0.04891	2.36	-0.27	0.8124
Intercept	1999g2	-0.3442	0.4466	16.8	-0.77	0.4515
Assignment	1999g2	0.01567	0.04891	2.36	0.32	0.7748
Intercept	1999g3	-0.2380	0.4466	16.8	-0.53	0.6011
Assignment	1999g3	0.008959	0.04891	2.36	0.18	0.8692
Intercept	1999h15	0.7776	0.4466	16.8	1.74	0.1000
Assignment	1999h15	-0.00208	0.04891	2.36	-0.04	0.9693
Intercept	1999j13	0.2514	0.4466	16.8	0.56	0.5809
Assignment	1999j13	-0.00095	0.04891	2.36	-0.02	0.9860
Intercept	1999j4	-0.5495	0.4466	16.8	-1.23	0.2356
Assignment	1999j4	0.003079	0.04891	2.36	0.06	0.9547
Intercept	1999j6	0.2416	0.4466	16.8	0.54	0.5956
Assignment	1999j6	-0.00567	0.04891	2.36	-0.12	0.9168
Intercept	1999p3	-0.2750	0.4466	16.8	-0.62	0.5463
Assignment	1999p3	-0.00385	0.04891	2.36	-0.08	0.9434
Intercept	1999p5	0.5162	0.4466	16.8	1.16	0.2640
Assignment	1999p5	0.009001	0.04891	2.36	0.18	0.8686
Intercept	1999p6	-0.4868	0.4466	16.8	-1.09	0.2911
Assignment	1999p6	0.02572	0.04891	2.36	0.53	0.6442

Intercept	1999p8	0.1140	0.4466	16.8	0.26	0.8015
Assignment	1999p8	0.01017	0.04891	2.36	0.21	0.8519
Intercept	1999t2	0.01897	0.4466	16.8	0.04	0.9666
Assignment	1999t2	-0.01675	0.04891	2.36	-0.34	0.7602
Intercept	1999u1	-0.3288	0.4466	16.8	-0.74	0.4718
Assignment	1999u1	0.01017	0.04891	2.36	0.21	0.8519
Intercept	1999u2	0.2825	0.4466	16.8	0.63	0.5356
Assignment	1999u2	0.005729	0.04891	2.36	0.12	0.9160
Intercept	1999u3	-0.08606	0.4466	16.8	-0.19	0.8495
Assignment	1999u3	0.01953	0.04891	2.36	0.40	0.7229
Intercept	1999u4	-0.4468	0.4466	16.8	-1.00	0.3313
Assignment	1999u4	0.002407	0.04891	2.36	0.05	0.9646
Intercept	1999w1	-0.6129	0.4466	16.8	-1.37	0.1881
Assignment	1999w1	0.01424	0.04891	2.36	0.29	0.7945
Intercept	1999w3	0.1550	0.4466	16.8	0.35	0.7328
Assignment	1999w3	-0.01321	0.04891	2.36	-0.27	0.8090
Intercept	1999w5	0.6402	0.4466	16.8	1.43	0.1701
Assignment	1999w5	-0.01211	0.04891	2.36	-0.25	0.8243
Intercept	1999w6	-0.7726	0.4466	16.8	-1.73	0.1020
Assignment	1999w6	0.05116	0.04891	2.36	1.05	0.3906
Intercept	2000b3	0.03865	0.4466	16.8	0.09	0.9321
Assignment	2000b3	0.02451	0.04891	2.36	0.50	0.6592
Intercept	2000b6	0.3488	0.4466	16.8	0.78	0.4457
Assignment	2000b6	-0.03065	0.04891	2.36	-0.63	0.5861
Intercept	2000b7	0.8056	0.4466	16.8	1.80	0.0893
Assignment	2000b7	-0.02748	0.04891	2.36	-0.56	0.6230
Intercept	2000b8	0.6679	0.4466	16.8	1.50	0.1534
Assignment	2000b8	0.01685	0.04891	2.36	0.34	0.7588
Intercept	2000b9	-0.2752	0.4466	16.8	-0.62	0.5461
Assignment	2000b9	-0.00349	0.04891	2.36	-0.07	0.9487
Intercept	2000c12	0.2533	0.4466	16.8	0.57	0.5782
Assignment	2000c12	-0.01280	0.04891	2.36	-0.26	0.8146
Intercept	2000c13	0.4540	0.4466	16.8	1.02	0.3238
Assignment	2000c13	-0.00099	0.04891	2.36	-0.02	0.9854
Intercept	2000c5	-1.2123	0.4466	16.8	-2.71	0.0149
Assignment	2000c5	0.008343	0.04891	2.36	0.17	0.8781
Intercept	2000e8	0.8047	0.4466	16.8	1.80	0.0896
Assignment	2000e8	-0.02159	0.04891	2.36	-0.44	0.6961
Intercept	2000f7	-0.1123	0.4466	16.8	-0.25	0.8045
Assignment	2000f7	-0.00074	0.04891	2.36	-0.02	0.9891
Intercept	2000j16	-0.1775	0.4466	16.8	-0.40	0.6960
Assignment	2000j16	0.008327	0.04891	2.36	0.17	0.8783
Intercept	2000j2	-0.1696	0.4466	16.8	-0.38	0.7090
Assignment	2000j2	-0.01475	0.04891	2.36	-0.30	0.7875
Intercept	2000j4	0.5086	0.4466	16.8	1.14	0.2708
Assignment	2000j4	-0.00441	0.04891	2.36	-0.09	0.9352
Intercept	2000j5	0.3223	0.4466	16.8	0.72	0.4804
Assignment	2000j5	-0.00593	0.04891	2.36	-0.12	0.9131
Intercept	2000k16	-0.2080	0.4466	16.8	-0.47	0.6474
Assignment	2000k16	0.02809	0.04891	2.36	0.57	0.6157
Intercept	2000k5	-0.5470	0.4466	16.8	-1.22	0.2376
Assignment	2000k5	-0.02131	0.04891	2.36	-0.44	0.6997
Intercept	2000k8	0.5752	0.4466	16.8	1.29	0.2152
Assignment	2000k8	-0.00236	0.04891	2.36	-0.05	0.9654
Intercept	2000m13	0.4368	0.4466	16.8	0.98	0.3420
Assignment	2000m13	-0.00243	0.04891	2.36	-0.05	0.9643
Intercept	2001c1	-0.7506	0.4466	16.8	-1.68	0.1114

Assignment	2001c1	0.02730	0.04891	2.36	0.56	0.6251
Intercept	2001c11	0.2915	0.4466	16.8	0.65	0.5228
Assignment	2001c11	0.01063	0.04891	2.36	0.22	0.8453
Intercept	2001c12	0.3296	0.4466	16.8	0.74	0.4707
Assignment	2001c12	-0.01506	0.04891	2.36	-0.31	0.7832
Intercept	2001c15	-1.0710	0.4466	16.8	-2.40	0.0284
Assignment	2001c15	-0.00003	0.04891	2.36	-0.00	0.9996
Intercept	2001c17	0.3558	0.4466	16.8	0.80	0.4367
Assignment	2001c17	-0.03524	0.04891	2.36	-0.72	0.5358
Intercept	2001c18	-0.8884	0.4466	16.8	-1.99	0.0632
Assignment	2001c18	0.01270	0.04891	2.36	0.26	0.8161
Intercept	2001c19	-0.1360	0.4466	16.8	-0.30	0.7645
Assignment	2001c19	-0.01342	0.04891	2.36	-0.27	0.8060
Intercept	2001c2	-0.5913	0.4466	16.8	-1.32	0.2032
Assignment	2001c2	0.02920	0.04891	2.36	0.60	0.6028
Intercept	2001c20	-0.5904	0.4466	16.8	-1.32	0.2039
Assignment	2001c20	0.05505	0.04891	2.36	1.13	0.3616
Intercept	2001c3	-1.3193	0.4466	16.8	-2.95	0.0090
Assignment	2001c3	0.03981	0.04891	2.36	0.81	0.4896
Intercept	2001f15	-0.4551	0.4466	16.8	-1.02	0.3227
Assignment	2001f15	0.01699	0.04891	2.36	0.35	0.7568
Intercept	2001h6	1.1690	0.4466	16.8	2.62	0.0182
Assignment	2001h6	-0.02395	0.04891	2.36	-0.49	0.6661
Intercept	2001i1	0.07415	0.4466	16.8	0.17	0.8701
Assignment	2001i1	-0.02184	0.04891	2.36	-0.45	0.6929
Intercept	2001i11	0.05225	0.4466	16.8	0.12	0.9083
Assignment	2001i11	-0.03754	0.04891	2.36	-0.77	0.5121
Intercept	2001i12	0.5468	0.4466	16.8	1.22	0.2377
Assignment	2001i12	-0.02628	0.04891	2.36	-0.54	0.6375
Intercept	2001i13	0.01078	0.4466	16.8	0.02	0.9810
Assignment	2001i13	0.04322	0.04891	2.36	0.88	0.4575
Intercept	2001i16	0.9648	0.4466	16.8	2.16	0.0455
Assignment	2001i16	-0.03592	0.04891	2.36	-0.73	0.5287
Intercept	2001i2	-0.01497	0.4466	16.8	-0.03	0.9737
Assignment	2001i2	0.000569	0.04891	2.36	0.01	0.9916
Intercept	2001i3	0.1973	0.4466	16.8	0.44	0.6643
Assignment	2001i3	-0.00983	0.04891	2.36	-0.20	0.8567
Intercept	2001i6	-2.2508	0.4466	16.8	-5.04	0.0001
Assignment	2001i6	0.05161	0.04891	2.36	1.06	0.3871
Intercept	2001i7	0.2889	0.4466	16.8	0.65	0.5265
Assignment	2001i7	0.02780	0.04891	2.36	0.57	0.6192
Intercept	2001i8	0.2046	0.4466	16.8	0.46	0.6528
Assignment	2001i8	-0.01433	0.04891	2.36	-0.29	0.7933
Intercept	2001i9	-0.3631	0.4466	16.8	-0.81	0.4277
Assignment	2001i9	0.01721	0.04891	2.36	0.35	0.7539
Intercept	2001m11	0.5245	0.4466	16.8	1.17	0.2567
Assignment	2001m11	-0.02173	0.04891	2.36	-0.44	0.6942
Intercept	2001m12	1.1613	0.4466	16.8	2.60	0.0188
Assignment	2001m12	-0.02447	0.04891	2.36	-0.50	0.6596
Intercept	2001m14	0.5189	0.4466	16.8	1.16	0.2616
Assignment	2001m14	-0.00437	0.04891	2.36	-0.09	0.9358
Intercept	2001m3	0.3950	0.4466	16.8	0.88	0.3890
Assignment	2001m3	0.002615	0.04891	2.36	0.05	0.9615
Intercept	2001m4	0.6047	0.4466	16.8	1.35	0.1937
Assignment	2001m4	-0.00954	0.04891	2.36	-0.19	0.8609
Intercept	2001m5	0.2304	0.4466	16.8	0.52	0.6127
Assignment	2001m5	-0.00783	0.04891	2.36	-0.16	0.8855

Intercept	2001m7	0.7501	0.4466	16.8	1.68	0.1116
Assignment	2001m7	-0.02660	0.04891	2.36	-0.54	0.6336
Intercept	2001q10	0.3245	0.4466	16.8	0.73	0.4775
Assignment	2001q10	-0.02628	0.04891	2.36	-0.54	0.6374
Intercept	2001r8	0.2736	0.4466	16.8	0.61	0.5484
Assignment	2001r8	-0.01673	0.04891	2.36	-0.34	0.7604
Intercept	2001s1	-0.5242	0.4466	16.8	-1.17	0.2569
Assignment	2001s1	0.000334	0.04891	2.36	0.01	0.9951
Intercept	2001s11	-0.2904	0.4466	16.8	-0.65	0.5243
Assignment	2001s11	-0.04058	0.04891	2.36	-0.83	0.4821
Intercept	2001s12	-0.4596	0.4466	16.8	-1.03	0.3180
Assignment	2001s12	-0.00114	0.04891	2.36	-0.02	0.9832
Intercept	2001s13	0.1187	0.4466	16.8	0.27	0.7936
Assignment	2001s13	-0.02920	0.04891	2.36	-0.60	0.6028
Intercept	2001s14	0.8471	0.4466	16.8	1.90	0.0752
Assignment	2001s14	-0.01228	0.04891	2.36	-0.25	0.8220
Intercept	2001s15	0.009684	0.4466	16.8	0.02	0.9830
Assignment	2001s15	0.02472	0.04891	2.36	0.51	0.6566
Intercept	2001s2	-0.5830	0.4466	16.8	-1.31	0.2094
Assignment	2001s2	0.01796	0.04891	2.36	0.37	0.7438
Intercept	2001s3	0.08654	0.4466	16.8	0.19	0.8487
Assignment	2001s3	0.007631	0.04891	2.36	0.16	0.8884
Intercept	2001s4	-0.9935	0.4466	16.8	-2.22	0.0402
Assignment	2001s4	0.03062	0.04891	2.36	0.63	0.5865
Intercept	2001s5	-0.3049	0.4466	16.8	-0.68	0.5042
Assignment	2001s5	0.01420	0.04891	2.36	0.29	0.7951
Intercept	2001s6	-1.5104	0.4466	16.8	-3.38	0.0036
Assignment	2001s6	0.02455	0.04891	2.36	0.50	0.6587
Intercept	2001s7	0.5843	0.4466	16.8	1.31	0.2084
Assignment	2001s7	0.01144	0.04891	2.36	0.23	0.8339
Intercept	2001s8	-0.7383	0.4466	16.8	-1.65	0.1169
Assignment	2001s8	0.07039	0.04891	2.36	1.44	0.2683
Intercept	2001s9	0.008021	0.4466	16.8	0.02	0.9859
Assignment	2001s9	0.001507	0.04891	2.36	0.03	0.9778

Type 3 Tests of Fixed Effects

Effect	Num DF	Den DF	F Value	Pr > F
Assignment	1	109	162.28	<.0001

C.18 RANDOM COEFFICIENT MIXED MODELS BY ASSIGNMENT FOR (PSPB, C++, ALLTEN)

Solution for Fixed Effects

Effect	Estimate	Standard Error	DF	t Value	Pr > |t|
Intercept	3.3630	0.1480	44	22.72	<.0001
Assignment	-0.1640	0.02135	44	-7.68	<.0001

Solution for Random Effects

Effect	Student	Estimate	Std Err Pred	DF	t Value	Pr > |t|
Intercept	1995p6	0.5800	0.3781	2.84	1.53	0.2277
Assignment	1995p6	0.03588	0.04814	1	0.75	0.5923
Intercept	1998i18	0.3479	0.3781	2.84	0.92	0.4289
Assignment	1998i18	0.05378	0.04814	1	1.12	0.4648
Intercept	1998x12	0.5846	0.3781	2.84	1.55	0.2249
Assignment	1998x12	0.02549	0.04814	1	0.53	0.6900
Intercept	1999h11	-0.7682	0.3781	2.84	-2.03	0.1404
Assignment	1999h11	-0.01702	0.04814	1	-0.35	0.7837
Intercept	1999h13	-0.2564	0.3781	2.84	-0.68	0.5489
Assignment	1999h13	0.01641	0.04814	1	0.34	0.7909
Intercept	1999j5	-0.2180	0.3781	2.84	-0.58	0.6069
Assignment	1999j5	-0.01303	0.04814	1	-0.27	0.8318
Intercept	1999t4	0.3500	0.3781	2.84	0.93	0.4265
Assignment	1999t4	-0.01213	0.04814	1	-0.25	0.8428
Intercept	1999v1	-0.09071	0.3781	2.84	-0.24	0.8267
Assignment	1999v1	-0.00098	0.04814	1	-0.02	0.9871
Intercept	1999v5	0.1384	0.3781	2.84	0.37	0.7399
Assignment	1999v5	-0.00616	0.04814	1	-0.13	0.9189
Intercept	2000b10	-0.1856	0.3781	2.84	-0.49	0.6589
Assignment	2000b10	0.03562	0.04814	1	0.74	0.5945
Intercept	2000c10	-0.7509	0.3781	2.84	-1.99	0.1465
Assignment	2000c10	-0.05869	0.04814	1	-1.22	0.4374
Intercept	2000c14	-0.5670	0.3781	2.84	-1.50	0.2358
Assignment	2000c14	-0.00413	0.04814	1	-0.09	0.9455
Intercept	2000c2	-0.01920	0.3781	2.84	-0.05	0.9629
Assignment	2000c2	0.01193	0.04814	1	0.25	0.8454
Intercept	2000c3	0.02136	0.3781	2.84	0.06	0.9587
Assignment	2000c3	-0.02114	0.04814	1	-0.44	0.7366
Intercept	2000c7	-0.1556	0.3781	2.84	-0.41	0.7098
Assignment	2000c7	-0.04114	0.04814	1	-0.85	0.5498
Intercept	2000c9	0.3712	0.3781	2.84	0.98	0.4024
Assignment	2000c9	0.02466	0.04814	1	0.51	0.6987
Intercept	2000e13	0.1953	0.3781	2.84	0.52	0.6430
Assignment	2000e13	0.004683	0.04814	1	0.10	0.9383
Intercept	2000e2	0.3081	0.3781	2.84	0.81	0.4779
Assignment	2000e2	0.007018	0.04814	1	0.15	0.9079
Intercept	2000e4	-0.08446	0.3781	2.84	-0.22	0.8384
Assignment	2000e4	-0.00564	0.04814	1	-0.12	0.9258
Intercept	2000e9	-0.08929	0.3781	2.84	-0.24	0.8293
Assignment	2000e9	0.01385	0.04814	1	0.29	0.8217
Intercept	2000f14	0.1516	0.3781	2.84	0.40	0.7168
Assignment	2000f14	-0.01205	0.04814	1	-0.25	0.8438
Intercept	2000g11	0.2069	0.3781	2.84	0.55	0.6244
Assignment	2000g11	-0.00288	0.04814	1	-0.06	0.9619
Intercept	2000g3	0.2055	0.3781	2.84	0.54	0.6265
Assignment	2000g3	-0.01967	0.04814	1	-0.41	0.7531
Intercept	2000g8	1.0320	0.3781	2.84	2.73	0.0766
Assignment	2000g8	0.03943	0.04814	1	0.82	0.5631
Intercept	2000j10	0.1985	0.3781	2.84	0.52	0.6379
Assignment	2000j10	0.009477	0.04814	1	0.20	0.8763
Intercept	2000j11	-0.2961	0.3781	2.84	-0.78	0.4937
Assignment	2000j11	0.01290	0.04814	1	0.27	0.8333
Intercept	2000j12	-0.7348	0.3781	2.84	-1.94	0.1525
Assignment	2000j12	-0.03263	0.04814	1	-0.68	0.6208

Intercept	2000j13	-0.1356	0.3781	2.84	-0.36	0.7450
Assignment	2000j13	0.01998	0.04814	1	0.41	0.7496
Intercept	2000j14	-0.3197	0.3781	2.84	-0.85	0.4632
Assignment	2000j14	0.01214	0.04814	1	0.25	0.8428
Intercept	2000j15	-0.5879	0.3781	2.84	-1.55	0.2230
Assignment	2000j15	0.01312	0.04814	1	0.27	0.8306
Intercept	2000j18	0.04799	0.3781	2.84	0.13	0.9075
Assignment	2000j18	-0.00923	0.04814	1	-0.19	0.8794
Intercept	2000j7	0.07687	0.3781	2.84	0.20	0.8526
Assignment	2000j7	0.01244	0.04814	1	0.26	0.8390
Intercept	2000j8	0.3068	0.3781	2.84	0.81	0.4796
Assignment	2000j8	-0.02673	0.04814	1	-0.56	0.6773
Intercept	2001k11	0.2850	0.3781	2.84	0.75	0.5086
Assignment	2001k11	0.03531	0.04814	1	0.73	0.5972
Intercept	2001m1	-1.1960	0.3781	2.84	-3.16	0.0548
Assignment	2001m1	-0.03697	0.04814	1	-0.77	0.5831
Intercept	2001m13	0.3766	0.3781	2.84	1.00	0.3964
Assignment	2001m13	0.02745	0.04814	1	0.57	0.6701
Intercept	2001q3	0.1724	0.3781	2.84	0.46	0.6810
Assignment	2001q3	-0.00817	0.04814	1	-0.17	0.8930
Intercept	2001q4	0.1031	0.3781	2.84	0.27	0.8038
Assignment	2001q4	-0.02163	0.04814	1	-0.45	0.7312
Intercept	2001q5	-0.3091	0.3781	2.84	-0.82	0.4767
Assignment	2001q5	-0.04197	0.04814	1	-0.87	0.5436
Intercept	2001q6	0.2436	0.3781	2.84	0.64	0.5678
Assignment	2001q6	-0.00915	0.04814	1	-0.19	0.8804
Intercept	2001q7	-0.3390	0.3781	2.84	-0.90	0.4395
Assignment	2001q7	-0.00619	0.04814	1	-0.13	0.9186
Intercept	2001r3	0.2718	0.3781	2.84	0.72	0.5269
Assignment	2001r3	0.02445	0.04814	1	0.51	0.7008
Intercept	2001r4	0.06026	0.3781	2.84	0.16	0.8840
Assignment	2001r4	0.000912	0.04814	1	0.02	0.9879
Intercept	2001t10	0.1789	0.3781	2.84	0.47	0.6702
Assignment	2001t10	0.01040	0.04814	1	0.22	0.8645
Intercept	2001t7	0.2885	0.3781	2.84	0.76	0.5039
Assignment	2001t7	-0.03999	0.04814	1	-0.83	0.5587

Type 3 Tests of Fixed Effects

Effect	Num DF	Den DF	F Value	Pr > F
Assignment	1	44	59.00	<.0001

C.19 DATA FOR TSP1

Obs	Module	Pgmr	DLDRR	ddDLDR	NDf DLDR	dreDLDR	DLDIR	ddDLDI	NDf DLDI	dreDLDI
1	M1	D	11.0092	27.523	25	0.12000	168.807	183.486	22	0.90909
2	M2	D	1.5598	0.000	69	0.00000	37.262	48.527	69	0.81159
3	M3	C	17.1359	2.448	60	0.03333	49.327	41.616	58	0.58621
4	M4	C	7.3260	0.000	52	0.00000	83.150	161.172	52	0.84615
5	M5	B	87.5000	100.000	8	0.50000	175.000	100.000	4	1.00000
6	M6	B	34.4086	32.258	22	0.13636	191.398	204.301	19	1.00000

7	M7	C	0.0000	19.608	15	0.20000	51.634	65.359	12	0.83333
8	M8	A	6.7708	0.000	42	0.00000	108.333	130.208	42	0.59524
9	M10	E	7.1429	10.204	12	0.16667	95.408	45.918	10	0.90000
10	M11	E	27.8846	0.000	9	0.00000	63.462	67.308	9	0.77778
11	M12	B	0.0000	0.000	1	0.00000	0.000	0.000	1	0.00000
12	M13	E	0.0000	0.000	0	.	0.000	0.000	0	.
13	M14	E	0.0000	0.000	0	.	96.923	0.000	0	.
14	M15	C	0.0000	0.000	0	.	0.000	0.000	0	.
15	M16	C	0.0000	0.000	0	.	0.000	0.000	0	.

Obs	Module	Pgmr	CRR	ddCR	NDf CR	dreCR	CIR	ddCI	NDf CI	dreCI
1	M1	D	14.6789	36.697	16	0.25000	16.5138	18.349	4	0.50000
2	M2	D	12.7383	25.997	99	0.30303	2.4263	28.596	46	0.71739
3	M3	C	4.0392	8.568	33	0.21212	3.7944	0.000	26	0.00000
4	M4	C	18.3150	18.315	13	0.38462	3.2967	0.000	8	0.00000
5	M5	B	62.5000	100.000	10	0.40000	22.5000	50.000	4	0.50000
6	M6	B	26.8817	21.505	13	0.15385	17.2043	86.022	10	0.80000
7	M7	C	13.0719	39.216	7	0.85714	3.2680	0.000	1	0.00000
8	M8	A	33.3333	36.458	20	0.35000	13.5417	0.000	6	0.00000
9	M10	E	5.1020	0.000	4	0.00000	11.2245	20.408	4	1.00000
10	M11	E	20.1923	0.000	12	0.00000	11.5385	115.385	12	1.00000
11	M12	B	14.8810	17.857	29	0.10345	9.5238	119.048	22	0.90909
12	M13	E	15.7895	0.000	2	0.00000	10.5263	52.632	2	1.00000
13	M14	E	1.5385	0.000	3	0.00000	10.7692	46.154	3	1.00000
14	M15	C	81.6327	0.000	0	.	4.0816	0.000	0	.
15	M16	C	57.1429	42.857	3	1.00000	4.2857	0.000	0	.

BIBLIOGRAPHY

Abbott, J.J. "Requirements Size Units: A Simple Software Size Measure." In *Proceedings of the Ninth International Conference on Software Quality in Cambridge, Massachusetts, October 4-6, 1999*, 13-26.

Ackerman, A.F. "Software Inspections and the Cost Effective Production of Reliable Software." In *Software Engineering*, ed. M. Dorfman and R.H. Thayer, 235-255. Los Alamitos, California: IEEE Computer Society Press, 1997.

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

Aitchison, J., and J.A.C. Brown. *The Lognormal Distribution.* London: Cambridge University Press, 1957.

Akiyama, F. "An Example of Software System Debugging." In *Proceedings of IFIP Congress 71 in Ljubljana, Yugoslavia, August 23-28, 1971*, 353-359.

Akiyama, Y., H. Fujita, T. Ohkubo, and A. Tominaga. "Another Secret of the Mythical Man-Month for Successful Project Planning." In *Proceedings of ProMAC2002: An International Conference on Project Management in Singapore, July 31 - August 2, 2002*, 1-8.

Albrecht, A. J., and J. Gaffney. "Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation." *IEEE Transactions on Software Engineering* 9, no. 6 (June 1983): 639-648.

American Institute of Aeronautics and Astronautics. *Recommended Practice for Software Reliability* (ANSI/AIAA R-013), 1992.

Austin, R.D. *Measuring and Managing Performance in Organizations*. New York: Dorset House Publishing, 1996.

Bach, J. "The Immaturity of the CMM." *American Programmer* 7, no. 9 (September 1994): 13-18.

Baker, F.T. "Chief Programmer Team Management of Production Programming." *IBM Systems Journal* 11, no. 1 (1972): 56-73.

Baker, F.T., and H.D. Mills. “Chief Programmer Teams.” *Datamation* 19, no. 12 (December 1973): 58-61.

Banker, R. D., S. M. Datar, and C. F. Kemerer. “Factors Affecting Software Maintenance Productivity: An Exploratory Study.” In *Proceedings of the 8th International Conference on Information Systems (ICIS) in Pittsburgh, Pennsylvania, December 1987*, 160-175.

_____. “A Model to Evaluate Variables Impacting Productivity on Software Maintenance Projects.” *Management Science*, 37, 1 (January 1991): 1-18.

Basili, V.R., and B.T. Perricone. “Software Errors and Complexity: An Empirical Investigation.” *Communications of the ACM* 27, no. 1 (January 1984): 42-52.

Beer, M., R.A. Eisenstat, and B. Spector. “Why Change Programs Don’t Produce Change.” *Harvard Business Review* 68, no. 6 (November/December 1990): 158-166.

Besselman, J., A. Arora, and P. Larkey. “A Feasible Reform of Software Procurement Using Software Process Improvement.” In *Proceedings of the 1995 Acquisition Research Symposium in Washington, DC*, 79-93.

Biffl, S. “Using Inspection Data for Defect Estimation.” *IEEE Software* 17, no. 6 (November/December 2000): 36-43.

Biffl, S., and W.J. Gutjahr. “Using a Reliability Growth Model to Control Software Inspection.” *Empirical Software Engineering* 7, no. 3 (September 2002): 257-284.

Boehm, B. *Software Engineering Economics*. Englewood Cliffs, New Jersey: Prentice-Hall, 1981.

Boehm, B., C. Abts, A.W. Brown, S. Chulani, B.K. Clark, E. Horowitz, R. Madachy, D. Reifer, and B. Steece. *Software Cost Estimation With COCOMO II*. Upper Saddle River, New Jersey: Prentice Hall, 2000.

Boehm, B., and R. Turner. *Balancing Agility and Discipline: A Guide for the Perplexed*. Boston: Addison-Wesley, 2004.

Bourgeois, K.V. “Process Insights from a Large-Scale Software Inspection Data Analysis.” *Crosstalk: the Journal of Defense Software Engineering* 9, no. 10 (October 1996): 17-22.

Bowers, C.A., J.A. Pharmer, and E. Salas. “When Member Homogeneity is Needed in Work Teams.” *Small Group Research* 31, no. 3 (June 2000): 305-326.

Brassard, M., and D. Ritter. *The Memory Jogger II*. Methuen, Massachusetts: GOAL/QPC, 1994.

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

Briand, L.C., K. El Emam, B.G. Freimut, O. Laitenberger. “A Comprehensive Evaluation of Capture-Recapture Models for Estimating Software Defect Content.” *IEEE Transactions on Software Engineering* 26, no. 6 (June 2000), p. 518-540.

Britz, G., D. Emerling, L. Hare, R. Hoerl, and J. Shade. “How to Teach Others to Apply Statistical Thinking.” *ASQ Quality Progress* 30, no. 6 (June 1997): 67-79.

Brocklehurst, S., and B. Littlewood. “Techniques for Prediction Analysis and Recalibration.” In *Handbook of Software Reliability Engineering*, ed. M.R. Lyu, 119-166. Los Alamitos, California: IEEE Computer Society Press, 1996.

Brodbeck, F.C., and T. Greitemeyer. “Effects of Individual versus Mixed Individual and Group Experience in Rule Induction on Group Member Learning and Group Performance.” *Journal of Experimental Social Psychology* 36, no. 6 (November 2000): 621-648.

Brooks, F.P., Jr. *The Mythical Man-Month: Essays on Software Engineering Anniversary Edition*. Boston: Addison-Wesley Publishing, 1995.

Campbell, M.D. “Methods, Tools, and Metrics for Performing the Empirical Analysis of Selected Traits of Individual Software Developers.” Ph.D. diss., University of South Carolina, 1999.

Carleton, A.D., M.C. Paulk, J. Barnard, A. Burr, D. Card, B. Curtis, A. Heijstek, B. Hirsh, T. Keller, S. Meade, and G. Wigle. “Panel Discussion: Can Statistical Process Control Be Usefully Applied to Software?” In *Proceedings of the 11th Software Engineering Process Group (SEPG) Conference in Atlanta, March 8-11, 1999*.

Chillarege, R., and I. Bhandari. “Orthogonal Defect Classification - A Concept for In-Process Measurements.” *IEEE Software* 18, no. 11 (November 1992): 943-955.

Compton, B.T., and C. Withrow. “Prediction and Control of Ada Software Defects.” *The Journal of Systems and Software* 12, no. 3 (July 1990): 199-207.

Constantine, L.L. *Constantine on Peopleware*. Englewood Cliffs, New Jersey: Yourdon Press Computing Series, 1995.

Cockburn, A. *Crystal Clear: A Human-Powered Methodology for Small Teams*. Boston: Addison-Wesley Publishing, 2004.

Cook, C. R., and A. Roesch. “Real-time Software Metrics.” *Journal of Systems and Software* 24, no. 3 (March 1994): 223-237.

Criscione, M., J. Ferree, and D. Porter. “Predicting Software Errors and Defects.” In *Proceedings of the 2001 Applications of Software Measurement in San Diego, February 12-16, 2001*, 269-280.

Curtis, B. “Substantiating Programmer Variability.” *Proceedings of the IEEE* 69, no. 7 (July 1981): 846.

_____. "The Impact of Individual Differences in Programmers." In *Working With Computers: Theory Versus Outcome*, ed. G.C. van der Veer, 279-294. London: Academic Press, 1988.

Curtis, B., H. Krasner, and N. Iscoe. "A Field Study of the Software Design Process for Large Systems." *Communications of the ACM* 31, no. 11 (November 1988): 1268-1287.

Curtis, B., W.E. Hefley, and S.A. Miller. *People Capability Maturity Model*. Reading, Massachusetts: Addison-Wesley, 2001.

Cusumano, M, A. MacCormack, C. Kemerer, and B. Crandall. "Software Development Worldwide: The State of the Practice." *IEEE Software* 20, no. 6 (November/December 2003): 28-34.

Das, N. "Study on Implementing Control Chart Assuming Negative Binomial Distribution with Varying Sample Size in a Software Industry." *ASQ Software Quality Professional* 6, no. 1 (December 2003): 38-39.

Davis, A.M. "Software Life Cycle Models." In *Software Engineering Project Management, Second Edition*, ed. R.H. Thayer, 105-114. Los Alamitos, California: IEEE Computer Society Press, 1997.

DeMarco, T., and T. Lister. *Peopleware, Second Edition.* New York: Dorset House, 1999.

_____. *Waltzing with Bears: Managing Risks on Software Projects.* New York: Dorset House, 2003.

Deming, W.E. *Out of the Crisis* (Cambridge: MIT Center for Advanced Engineering Study, 1986.

Devnani-Chulani, S. "Bayesian Analysis of Software Cost and Quality Models." Ph.D. diss., University of Southern California, 1999.

Dijkstra, E. "Structured Programming." *Classics in Software Engineering*, E.N. Yourdon (ed), (New York: Yourdon Press, 1979.

Duncan, T.E., S.C. Duncan, L.A. Strycker, F. Li, and A. Alpert. *An Introduction to Latent Variable Growth Curve Modeling*. Mahwah, New Jersey: Lawrence Erlbaum Associates, 1999.

Eick, S.G., C.R. Loader, M.D. Long, S.A.V. Wiel, and L.G. Votta. "Estimating Software Fault Content Before Coding." In *Proceedings of the 14th International Conference on Software Engineering in Melbourne, Australia, May 1992*.

El Emam, K., and O. Laitenberger. "Evaluating Capture-Recapture Models With Two Inspectors." *IEEE Transactions on Software Engineering* 27, no. 9 (September 2001): 851-864.

Endres, A., and D. Rombach. *A Handbook of Software and Systems Engineering: Empirical Observations, Laws, and Theories*. Boston: Addison Wesley, 2003.

Evanco, W.M., and R. Lacovava. "A Model-Based Framework for the Integration of Software Metrics." *The Journal of Systems and Software* 26, 1994, pp. 77-86.

Fagan, M.E. "Design and Code Inspections to Reduce Errors in Program Development." *IBM Systems Journal* 15, no. 3 (1976): 182-211.

_____. "Advances in Software Inspections." *IEEE Transactions on Software Engineering* 12, no. 7 (July 1986): 744-751.

Fenton, N., P. Krause, and M. Neil. "Software Measurement: Uncertainty and Causal Modeling." *IEEE Software* 19, no. 5 (July/August 2002): 116-122.

Fenton, N., and M. Neil. "A Critique of Software Defect Prediction Models." *IEEE Transactions on Software Engineering* 25, no. 5 (September/October 1999): 675-689.

_____. "Software Metrics: Successes, Failures, and New Directions." *The Journal of Systems and Software* 47, no. 2-3 (July 1999): 149-157.

Fenton, N., and N. Ohlsson. "Quantitative Analysis of Faults and Failures in a Complex Software System." *IEEE Transactions on Software Engineering* 26, no. 8 (August 2000): 797-814.

Fenton, N., S.L. Pfleeger, and R. Glass. "Science and Substance: A Challenge to Software Engineers." *IEEE Software* 11, no. 4 (July 1994): 86-95.

Ferdinand, A.E. "A Theory of System Complexity." *International Journal of General Systems* 1 (1974): 19-33.

Ferguson, P., W.S. Humphrey, S. Khajenoori, S. Macke, and A. Matvya. "Results of Applying the Personal Software Process." *IEEE Computer* 30, no. 5 (May 1997): 24-31.

Florac, W.A. "Software Quality Measurement: A Framework for Counting Problems and Defects." Pittsburgh: Software Engineering Institute, Carnegie Mellon University, 1992. CMU/SEI-92-TR-22.

Florac, W.A., and A.D. Carleton. *Measuring the Software Process: Statistical Process Control for Software Process Improvement*. Reading, Massachusetts: Addison-Wesley, 1999.

Florac, W.A., A.D. Carleton, and J. Barnard. "Statistically Managing the Software Process." *IEEE Software* 17, no. 4 (July/August 2000): 97-106.

Freedman, D., and G.M. Weinberg. *Handbook of Walkthroughs, Inspections, and Technical Reviews, Third Edition*. New York: Dorset House, 1990.

Gaffney, J.R. “Estimating the Number of Faults in Code.” *IEEE Transactions on Software Engineering* 10, no. 4 (July 1984): 459-465.

Garvin, D.A. “Competing on the Eight Dimensions of Quality.” *Harvard Business Review* 65, no. 6 (November/December 1987): 101-109.

Gibbs, W.W. “Software’s Chronic Crisis.” *Scientific American*, (September 1994): 86-95.

Gill, G. K., and C. F. Kemerer. “Cyclomatic Complexity Density and Software Maintenance Productivity.” *IEEE Transactions on Software Engineering*, 17, no. 12, (December 1991): 1284-1288.

Glass, R.L. *Software Creativity*. Englewood Cliffs, NJ: Prentice Hall, 1995.

_____. “Inspections - Some Surprising Findings.” *Communications of the ACM* 42, no. 4 (April 1999): 17-19.

_____. *Facts and Fallacies of Software Engineering*. Boston: Addison Wesley, 2004.

Goethert, W.B., E.K. Bailey, and M.B. Busby. “Software Effort and Schedule Measurement: A Framework for Counting Staff-Hours and Reporting Schedule Information.” Pittsburgh: Software Engineering Institute, Carnegie Mellon University, 1992. CMU/SEI-92-TR-21.

Goldenson, D.R., and J.D.Herbsleb. “After the Appraisal: A Systematic Survey of Process Improvement, its Benefits, and Factors that Influence Success.” Pittsburgh: Software Engineering Institute, Carnegie Mellon University, 1995. CMU/SEI-95-TR-009.

Grady, R. B., and T. van Slack. “Key Lessons in Achieving Widespread Inspection Use.” *IEEE Software* 11, no. 4 (July/August 1994): 46-57.

Graves, T.L., A.F. Karr, J.S. Marron, and H. Siy. “Predicting Fault Incidence Using Software Change History.” *IEEE Transactions on Software Engineering* 26, no. 7 (July 2000): 653-661.

Hahn, G.J. “How Abnormal Is Normality?” *Journal of Quality Technology* 3, no. 1 (January 1971): 18-22.

Hahn, G.J., and W.Q. Meeker. “Assumptions for Statistical Inference.” *The American Statistician* 47, no. 1 (February 1993): 1-11.

Hall, P., and A. Nixon. “Fagan Inspection Data Analysis: A Practical Approach.” In *Proceedings of SPIRE 2000 (Software Process Improvement Realisation and Evaluation Conference) in Hinckley, England, October 2-4 2000*.

Halstead, M.H. *Elements of Software Science* (New York: Elsevier Computer Science Library, 1977.

Hare, L.B., R.W. Hoerl, J.D. Hromi, and R.D. Snee. “The Role of Statistical Thinking in Management.” *ASQ Quality Progress* 28, no. 2 (February 1995): 53-60.

Hatton, L. “Reexamining the Fault Density – Component Size Connection.” *IEEE Software* 14, no. 2 (March/April 1997): 89-97.

Hayes, W. “The Status of the PSP Data Set.” Pittsburgh: Software Engineering Institute, Carnegie Mellon University, 1996.

_____. “Using a Personal Software Process to Improve.” In *Proceedings of the Fifth International Software Metrics Symposium in Bethesda, Maryland, 1998*, 61-71.

Hayes, W. and J.W. Over. “The Personal Software Process (PSP): An Empirical Study of the Impact of PSP on Individual Engineers.” Pittsburgh: Software Engineering Institute, Carnegie Mellon University, 1997. CMU/SEI-97-TR-001.

Hemdal, J.E., and R.L. Galen. “The Personal Software Process -- A Few Unexpected Lessons.” In *Proceedings of Applications of Software Measurement in San Jose, Californai, March 6-10, 2000*, 1-22.

Highsmith, J.A. *Adaptive Software Development: A Collaborative Approach to Managing Complex Systems*. New York: Dorset House Publishing, 2000.

Hilburn, T.B., and W.S. Humphrey. “Teaching Teamwork.” *IEEE Software* 19, no. 5, (September/October 2002): 72-77.

Hirmanpour, I., and J. Schofield. “Defect Management Through the Personal Software Process.” *Crosstalk: The Journal of Defense Software Engineering* 16, no. 9 (September 2003): 17-20.

Hogg, R.V., and J. Ledolter. *Applied Statistics for Engineers and Physical Scientists*. New York: Macmillan, 1992.

Holmes, J.S. “Optimizing the Software Life Cycle.” *ASQ Software Quality Professional* 5, no. 4 (September 2003): 14-23.

Hou, L., and J.E. Tomayko. “Applying the Personal Software Process in CS1: An Experiment.” In *Proceedings of the Twenty-ninth SIGCSE Technical Symposium on Computer Science Education in Atlanta, February 25 - March 1, 1998*, 322-325.

Humphrey, W.S. “Characterizing the Software Process.” *IEEE Software* 5, no. 2 (March 1988): 73-79.

_____. *Managing the Software Process*. Boston: Addison-Wesley, 1989.

_____. “CASE Planning and the Software Process.” Pittsburgh: Software Engineering Institute, Carnegie Mellon University, 1989. CMU/SEI-89-TR-26.

_____. *A Discipline for Software Engineering*. Reading, Massachusetts: Addison-Wesley, 1995.

_____. "Using a Defined and Measured Personal Software Process." *IEEE Software* 13, no. 3 (May 1996): 72-88.

_____. *Introduction to the Team Software Process*. Reading, Massachusetts: Addison-Wesley, 1999.

Institute of Electrical and Electronics Engineers. *IEEE Standard for Software Reviews and Audits* (ANSI/IEEE Std 1028), 1988.

Institute of Electrical and Electronics Engineers. *IEEE Standard Glossary of Software Engineering Terminology* (ANSI/IEEE Std 610), 1991.

Ishikawa, K. *Guide to Quality Control*. White Plains, New York: Kraus International Publications, 1986.

International Organization for Standardization. *Software Quality Characteristics* (ISO/IEC 9126), 1991.

JMP Statistics and Graphics Guide, Version 4. Cary, North Carolina: SAS Institute, 2000.

Johnson, N.L., and S. Kotz. *Discrete Distributions* (New York: John Wiley and Sons, 1969.

Johnson, P.M., and A.M. Disney. "The Personal Software Process: A Cautionary Case Study." *IEEE Software* 15, no. 6 (November/December 1998): 85-88.

_____. "A Critical Analysis of PSP Data Quality: Results from a Case Study." *Empirical Software Engineering* 4, no. 4 (1999): 317-349.

Jones, C. "Backfiring" or Converting Lines of Code Metrics Into Function Points, (Burlington, Massachusetts: Software Productivity Research, October 1995).

_____. "Software Benchmarking." *IEEE Computer* 28, no. 10 (October 1995): 102-103.

_____. "Software Estimating Rules of Thumb." *IEEE Computer* 29, no. 3 (March 1996): 116-118.

_____. *Applied Software Measurement, Second Edition*. New York: McGraw Hill, 1997.

Jose, A., N.K. Anju, and S.K. Pillai. "Closed-Loop Defect Removal Model Using Statistical Process Control." *ASQ Software Quality Professional* 3, no. 1 (December 2000): 39-47.

Judd, C. M., and G.H. McClellan. *Data Analysis: A Model-Comparison Approach.* San Diego: Harcourt Brace Jovanovich, 1989.

Kan, S.H. *Metrics and Models in Software Quality Engineering, Second Edition*. Boston: Addison-Wesley, 2003.

Keiller, P.A.R. “Improving the Predictive Performance of the Nonhomogeneous Poisson Process Software Reliability Growth Models.” Ph.D. diss., George Washington University, 1995.

Kemerer, C. F. “Reliability of Function Points Measurement: A Field Experiment.” *Communications of the ACM* 36, no. 2 (February 1993): 85-97.

Khattree, R., and D.N. Naik. *Applied Multivariate Statistics with SAS Software*. Cary, North Carolina: SAS Publishing, 1999.

Khoshgoftaar, T.M., E.B. Allen, R. Halstead, G.P. Trio, and R.M. Flass. “Using Process History to Predict Software Quality.” *IEEE Software* 31, no. 4 (April 1998): 66-72.

Khoshgoftaar, T.M., and R.M. Szabo. “An Application of Zero-Inflated Poisson Regression for Software Fault Prediction.” In *Proceedings of the 12th International Symposium on Software Reliability Engineering in Hong Kong, November 27-30, 2001*, 66-73.

Kitchenham, B.A., L.M. Pickard, and S.J. Linkman. “An Evaluation of Some Design Metrics.” *Software Engineering Journal* 5, no. 1 (January 1990): 50-58.

Kramer, S.H. “When Are Two Heads Better Than One? The Role of Expertise and Task Difficulty in Individual, Statistical Group, and Interacting Group Problem Solving.” Ph.D. diss., Harvard University, 1998.

Krasner, H. “Self-Assessment Experience at Lockheed.” In *Proceedings of the Third Annual SEPG Workshop in Pittsburgh, November 7, 1990*.

_____. “The Cost of Software Quality (CoSQ): Empowering Improvement.” In *Proceedings of the 7th International Conference on Software Quality in Montgomery, Alabama, October 6-8, 1997*.

_____. “Accumulating the Body of Evidence for the Payoff of Software Process Improvement – 1997.” In *Software Process Improvement,* ed. R.B. Hunter and R.H. Thayer, 519-539. Los Alamitos, California: IEEE Computer Society, 2001.

Land, L.P.K. “Software Group Reviews and the Impact of Procedural Roles on Defect Detection Performance.” Ph.D. diss., University of New South Wales, 2002.

Leemis, L.M. *Reliability: Probabilistic Models and Statistical Methods.* Upper Saddle River, New Jersey: Prentice Hall, 1995.

Lehman, M.M., J.F. Ramil, P.D. Wernick, D.E. Perry, and W.M. Turski. “Metrics and Laws of Software Evolution – The Nineties View.” In *Elements of Software Process Assessment and Improvement*, ed. K. El Emam and N.H. Madhavji, 343-368. Los Alamitos: IEEE Computer Society Press, 1999.

Liberatore, R.L. “Performance and Efficiency of Individuals Charts in Applications to Obtain Statistical Control.” Ph.D. diss., University of Pittsburgh, 1995.

Lipke, W., and M. Jennings. “Software Project Planning, Statistics, and Earned Value.” *Crosstalk: The Journal of Defense Software Engineering* 13, no. 12 (December 2000): 10-14.

Lipow, M. “Number of Faults per Line of Code.” *IEEE Transactions on Software Engineering* 8, no. 4 (July 1982): 437-439.

Littell, R.C., G.A. Milliken, W.W. Stroup, and R.D. Wolfinger. *SAS System for Mixed Models*. Cary, North Carolina: SAS Publishing, 1996.

Lyu, M.R., ed. *Handbook of Software Reliability Engineering*. Los Alamitos, California: IEEE Computer Society Press, 1996.

MacCormack, A., C.F. Kemerer, M. Cusumano, and B. Crandall. “Trade-offs Between Productivity and Quality in Selecting Software Development Practices.” *IEEE Software* 20, no. 5 (September/October 2003): 83-84.

Mah, M. “Defect Metrics, Inspections, and Testing: Pay Me Now, or Pay Me Later.” *IT Metrics Strategy* 7, no. 1 (January 2001), p. 12.

Manley, J.H. *Rise Above the Rest* (Pittsburgh: Cathedral Publishing, 1998.

Mayer, A., and A.M. Sykes. “Statistical Methods for the Analysis of Software Metrics Data.” *Software Quality Journal* 1, no. 4, (December 1992): 209-223.

McAndrews, D.R. “The Team Software Process (TSP): An Overview and Preliminary Results of Using Disciplined Practices.” Pittsburgh: Software Engineering Institute, Carnegie Mellon University, 2000. CMU/SEI-2000-TR-015.

McCann, R.T. “How Much Code Inspection Is Enough?” *Crosstalk: The Journal of Defense Software Engineering* 14, no. 7 (July 2001): 9-12.

McGarry, F.E. “What Have We Learned in the Last Six Years: Measuring Software Development Technology.” In *Proceedings of the Seventh Annual Software Engineering Workshop in Greenbelt, Maryland, 1982*, 205-238.

Milliken, G.A., and D.E. Johnson. *Analysis of Messy Data, Volume I: Designed Experiments.* New York: Chapman & Hall, 1992.

Montgomery, D.C. *Introduction to Statistical Quality Control* (3rd edition; New York: John Wiley & Sons, 1996.

Montgomery, D.C., and G.C. Runger. *Applied Statistics and Probability for Engineers, Second Edition* (New York: John Wiley & Sons, 1999.

Morgan, P.M., and R.S. Tindale. “Group vs Individual Performance in Mixed-Motive Situations: Exploring an Inconsistency.” *Organizational Behavior and Human Decision Processes* 87, no. 1 (January 2002), p. 46.

Mullen, R.E. “The Lognormal Distribution of Software Failure Rates: Application to Software Reliability Growth Modeling.” In *Proceedings of the Ninth International Symposium on Software Reliability Engineering in Paderborn, Germany, November 4-7, 1998*, 124-133.

Musa, J.D., A. Iannino, and K. Okumoto. *Software Reliability: Measurement, Prediction, Application*. New York: McGraw-Hill, 1987.

National Institute of Standards and Technology. *The Economic Impacts of Inadequate Infrastructure for Software Testing,* May 2002.

Nelson, L. “Control Charts for Individual Measures.” *Journal of Quality Technology* 14, no. 3 (July 1982): 172-174.

Neter, J., M.H. Kutner, and C.J. Nachtscheim. *Applied Linear Statistical Models, Fourth Edition.* Chicago: Irwin, 1996.

Neufelder, A.M. “How to Measure the Impact of Specific Development Practices on Fielded Defect Density.” In *Proceedings of the 11th International Symposium on Software Reliability Engineering in San Jose, California, October 8-11, 2000,* 148-159.

Nikora, A.P. “Software System Defect Content Prediction from Development Process and Product Characteristics.” Ph.D. diss., University of Southern California, 1998.

Orr, J.M., P.R. Sackett, and C.L.Z. DuBois. (1991). “Outlier Detection and Treatment in Psychology: A Survey of Researcher Beliefs and an Empirical Illustration.” *Personnel Psychology* 44 (1991), 473- 486.

Osborne, J. “Notes on the Use of Data Transformations.” *Practical Assessment, Research & Evaluation* 8, no.6 (2002).

Ould, M.A. “CMM and ISO 9001.” *Software Process: Improvement and Practice* 2, no. 4 (December 1996):281-289.

Park, R.E. “Software Size Measurement: A Framework for Counting Source Statements.” Pittsburgh: Software Engineering Institute, Carnegie Mellon University, 1992. CMU/SEI-92-TR-20.

Parnas, D.L., and D.M. Weiss. “Active Design Reviews: Principles and Practices.” *The Journal of Systems and Software* 7, no. 4 (December 1987): 259-265.

Parsons, H.M. “What Happened at Hawthorne?” *Science* 183, no. 4128 (March 8, 1974): 922-932.

Paulk, M.C. “Toward Quantitative Process Management With Exploratory Data Analysis.” In *Proceedings of the Ninth International Conference on Software Quality in Cambridge, Massachusetts, October 4-6, 1999*, 35-42.

_____. “Models and Standards for Software Process Assessment and Improvement.” In *Software Process Improvement,* ed. R.B. Hunter and R.H. Thayer, 5-37. Los Alamitos, California: IEEE Computer Society, 2001.

Paulk, M.C. and M.B. Chrissis. “The November 1999 High Maturity Workshop.” Pittsburgh: Software Engineering Institute, Carnegie Mellon University, 2000. CMU/SEI-2000-SR-003.

Paulk, M.C., D.R. Goldenson, and D.M. White. “The 1999 Survey of High Maturity Organizations.” Pittsburgh: Software Engineering Institute, Carnegie Mellon University, 2000. CMU/SEI-2000-SR-002.

Paulk, M.C., C. Weber, B. Curtis, and M.B. Curtis. *The Capability Maturity Model: Guidelines for Improving the Software Process*. Reading, Massachusetts: Addison-Wesley, 1995).

Perry, D.E., A. Porter, M.W. Wade, L.G. Votta, and J. Perpich. “Reducing Inspection Interval in Large-Scale Software Development.” *IEEE Transactions on Software Engineering* 28, no. 7 (July 2002): 695-705.

Phipps, G. “Comparing Observed Bug and Productivity Rates for Java and C++.” *Software – Practice and Experience* 29, no. 4 (April 1999): 345-358.

Pierce, V.D. “Process Stability Analysis.” In *Accenture Process Improvement Conference, June 16-17, 2005*.

Porter, A.A., and P.M. Johnson. “Assessing Software Review Meetings: Results of a Comparative Analysis of Two Experimental Studies.” *IEEE Transactions on Software Engineering* 23, no. 3 (March 1997): 129-145.

Porter, A.A., and L.G. Votta. “What Makes Inspections Work?” *IEEE Software* 14, no. 6 (November/December 1997): 99-102.

_____. “Comparing Detection Methods for Software Requirements Inspections: A Replication Using Professional Subjects.” *Empirical Software Engineering* 3, no. 4 (December 1998): 355-379.

Porter, A.A., H. Siy, C. Toman and L.G. Votta. “An Experiment to Assess the Cost-Benefits of Code Inspections in Large Scale Software Development.” *IEEE Transactions on Software Engineering*, 23, no. 6 (June 1997): 329-346.

Porter, D. “Statistical Process Control (SPC) for Software Inspections.” In *Proceedings of the 2001 Applications of Software Measurement in San Diego, February 12-16, 2001*, 473-486.

Prechelt, L., and B. Unger. “An Experiment Measuring the Effects of Personal Software Process (PSP) Training.” *IEEE Transactions on Software Engineering* 27, no. 5 (May 2000): 465-472.

Putnam, L.H., and W. Myers. *Measures for Excellence* (Englewood Cliffs, New Jersey: Yourdon Press, 1992.

_____. *Industrial Strength Software: Effective Management Using Measurement* (Los Alamitos, California: IEEE Computer Society Press, 1997.

Pyzdek, T. "Process Control for Short and Small Runs." *ASQ Quality Progress* 26, no. 4 (April 1993): 51-60.

Radice, R.A. *High Quality Low Cost Software Inspections*. Andover, Massachusetts: Paradoxicon Publishing, 2002.

Raffo, D.M. "Modeling Software Processes Quantitatively and Assessing the Impact of Potential Process Changes on Process Performance." Ph.D. diss., Carnegie Mellon University, 1996.

Rawlings, J.O., S.G. Pantula, and D.A. Dickey. *Applied Regression Analysis, Second Edition* (New York: Springer-Verlag, 1998.

Rifkin, S. "The Business Case for Software Process Improvement." In *Proceedings of the Fifth SEPG National Meeting in Costa Mesa, California, April 26-29, 1993*.

Rocke, D.M. "Robust Control Charts." *Technometrics* 31, no. 2 (May 1988): 173-184.

Roes, K.C.B., R.J.M.M. Does, and Y. Schurink. "Shewhart-Type Control Charts for Individual Observations." *Journal of Quality Technology* 25, no. 3 (July 1993): 188-198.

Sackman, H., W.J. Erikson, and E.E. Grant. "Exploratory Experimental Studies Comparing Online and Offline Programming Performance." *Communications of the ACM* 11, no. 1 (January 1968): 3-11.

SAS Institute Inc. *SAS/STAT User's Guide, Version 6, Fourth Edition, Volume 1*. Cary, North Carolina: SAS Publishing, 1989.

Sawyer, S., and P.J. Guinan. "Software Development: Processes and Performance." *IBM Systems Journal* 37, no. 4 (1998): 552-569.

Schaefer, H. "Metrics for Optimal Maintenance Management." In *Proceedings of the Conference on Software Maintenance in Washington DC, 1995*, 114-119.

Schilling, E.G., and P.R. Nelson. "The Effect of Non-Normality on the Control Limits of $\bar{X}$ Charts." *Journal of Quality Technology* 8, no. 4 (October 1976): 183-188.

Schneberger, S.L. "Distributed Computing Environments: Effects on Software Maintenance Difficulty." *The Journal of Systems and Software* 37, 1997, pp. 101-116.

Scholtes, P.R., B.L. Joiner, and B.J. Streibel. *The TEAM Handbook, Second Edition*. Madison, Wisconsin: Oriel Incorporated, 1996.

Shaw, M. “Prospects for an Engineering Discipline of Software.” *IEEE Software* 7, no. 6 (November 1990): 15-24.

Shen, V.Y., T. Yu, S.M. Thebaut, and L.R. Paulsen. “Identifying Error-Prone Software – An Empirical Study.” *IEEE Transactions on Software Engineering* 11, no. 4 (April 1985): 317-323.

Sheppard, S.B., B. Curtis, P. Milliman, and T. Love. “Modern Coding Practices and Programmer Performance.” *IEEE Computer* 12, no. 12 (December 1979): 41-49.

Shewhart, W.A. *Economic Control of Quality of Manufactured Product.* New York: Van Nostrand, 1931.

_____. *Statistical Method from the Viewpoint of Quality Control.* Mineola, New York: Dover Publications, 1939.

Takahasi, M., and Y. Kamayachi. “An Empirical Study of a Model for Program Error Prediction.” In *Proceedings of the 8th International Conference on Software Engineering in London, August, 1985*, 330-336.

Tichy, W.F., P. Lukowicz, L. Prechelt, and E.A. Heinz. “Experimental Evaluation in Computer Science: A Quantitative Study.” *The Journal of Systems and Software* 28, no. 1 (January 1995): 9-18.

Votta, L. G. “Does Every Inspection Need a Meeting?” In *Proceedings of the First ACM SIGSOFT Symposium on the Foundations of Software Engineering in Los Angeles, CA, 1993*, 107-114.

Webb, D.R. “Managing Risk with TSP.” *Crosstalk: The Journal of Defense Software Engineering* 13, no. 6 (June 2000): 7-10.

Webb, D.R., and W.S. Humphrey. “Using the TSP on the TaskView Project.” *Crosstalk: The Journal of Defense Software Engineering* 12, no. 2 (February 1999): 3-10.

Weinberg, G.M. *The Psychology of Computer Programming: Silver Anniversary Edition.* New York: Dorset House, 1998.

Weller, E.F. “Lessons from Three Years of Inspection Data.” *IEEE Software* 10, no. 5 (September 1993): 38-45.

_____. “Practical Applications of Statistical Process Control.” *IEEE Software* 17, no. 3 (May/June 2000): 48-55.

Wesslen, A. “Improving Software Quality through Understanding and Early Estimations.” Ph.D. diss., Lund Institute of Technology, 1999.

_____. “A Replicated Empirical Study of the Impact of the Methods in the PSP on Individual Engineers.” *Empirical Software Engineering* 5, no. 2 (June 2000): 93-123.

Wheeler, D.J. *Normality and the Process Behavior Chart.* Knoxville, Tennessee: SPC Press, 2000.

_____. *Making Sense of Data: SPC for the Service Sector.* Knoxville, Tennessee: SPC Press, 2003.

Wheeler, D.J., and D.S. Chambers. *Understanding Statistical Process Control, Second Edition.* Knoxville, Tennessee: SPC Press, 1992.

Wheeler, D.J., and S.R. Poling. *Building Continual Improvement: A Guide for Business*. Knoxville, Tennessee: SPC Press, 1998.

Williams, K.D. "The Value of Software Improvement… Results! Results! Results!" In *Proceedings of SPIRE 1997 (Software Process Improvement Realisation and Evaluation Conference), June 4, 1997*.

Williams, L.A. "The Collaborative Software Process." Ph.D. diss., University of Utah, 2000.

Williams, L.A., R.R. Kessler, W. Cunningham, and R. Jeffries. "Strengthening the Case for Pair Programming." *IEEE Software* 17, no. 4 (July/August 2000): 19-25.

Williamson, E., and M.H. Bretherton. *Tables of the Negative Binomial Distribution*. New York: John Wiley and Sons, 1963.

Withrow, C. "Error Density and Size in Ada Software." *IEEE Software*, 7, no. 1 (January/February 1990): 26-30.

Wohlin, C. "Are Individual Differences in Software Development Performance Possible to Capture Using a Quantitative Survey?" *Empirical Software Engineering* 9, no. 3 (September 2004): 211-228.

Wohlin, C., and H. Petersson. "Defect Content Estimation for Two Reviewers." In *Proceedings of the 12th International Symposium on Software Reliability Engineering in Hong Kong, November 27-30, 2001*, 340-345.

Wohlin, C., and P. Runeson. "Defect Content Estimations from Review Data." In *Proceedings of the 20th International Conference on Software Engineering in Kyoto, Japan, April 19-25, 1998*, 400-409.

Wohlin, C., and A. Wesslen. "Understanding Software Defect Detection in the Personal Software Process." In *Proceedings of the Ninth International Symposium on Software Reliability in Paderborn, Germany, November 4-7, 1998*, 49-58.

Yang, Z., and J.C. Paradi. "DEA Evaluation of a Y2K Software Retrofit Program." *IEEE Transactions on Engineering Management* 51, no. 3 (August 2004): 279-287.

Zhang, X. "Software Reliability and Cost Models with Environmental Factors." Ph.D. diss., Rutgers, 1999.