Does Software Process Improvement Reduce the Severity of Defects? A Longitudinal Field Study

Harter, Donald E., Kemerer, Chris F., *Member, IEEE Computer Society* and Slaughter, Sandra A.

Abstract— As firms increasingly rely on information systems to perform critical functions the consequences of software defects can be catastrophic. Although the software engineering literature suggests that software process improvement can help to reduce software defects, the actual evidence is equivocal. For example, improved development processes may only remove the "easier" syntactical defects, while the more critical defects remain. Rigorous empirical analyses of these relationships have been very difficult to conduct due to the difficulties in collecting the appropriate data on real systems from industrial organizations. This field study analyzes a detailed data set consisting of 7,545 software defects that were collected on software projects completed at a major software firm. Our analyses reveal that higher levels of software process improvement significantly reduce the likelihood of high severity defects. In addition, we find that higher levels of process improvement are even more beneficial in reducing severe defects when the system developed is large or complex, but are less beneficial when requirements are ambiguous, unclear or incomplete. Our findings reveal the benefits and limitations of software process improvement for the removal of severe defects and suggest where investments in improving development processes may have their greatest effects.

Index Terms— software complexity, defect severity, requirements ambiguity, software process, CMM.

1 Introduction

Software defects have historically plagued the software industry. The National Institute of Standards and Technology (NIST) estimates that poor software quality costs U.S. businesses $59.5 billion per year, split between software developers ($21.2 billion) and users ($38.3 billion) [1]. Further, the consequences of software defects are growing as individuals, organizations and society increasingly rely on software systems for critical functions. As noted by Mann, "in the last 15 years alone, software defects have wrecked a European satellite launch, delayed the opening of the hugely expensive Denver airport for a year, destroyed a NASA Mars mission, killed four marines in a helicopter crash, induced a U.S. Navy ship to destroy a civilian airliner, and shut down ambulance systems in London, leading to as many as 30 deaths." [2]

How can software engineers improve this situation? There is some research suggesting that software process improvement efforts, such as the Capability Maturity Model (CMM) can help reduce defect rates. For example, respondents to a survey conducted by Herbsleb *et al.* reported that improvements in processes, as measured by the CMM, led to significant reductions in defects [3]. Subsequent studies [4], [5] found that investments in CMM process improvements resulted in significant improvements in both software quality and costs.

However, this prior work has not had the opportunity to examine whether and/or how software process improvement affects the *severity* of defects. Severity is a measure of the impact a defect has on a system and its users. Crosby identified the importance of defect severity as early as 1979 [6]. More recently, Jones re-emphasized the importance of measuring severity levels rather than simply identifying the number of defects [7]. An examination of defect severity has the potential to contribute to the research on software quality because not all software errors are equal. While some defects are merely cosmetic, others can be catastrophic with potentially tragic consequences. Two recently documented examples are:

- A software flaw in GE's energy management system contributed to the devastating scope of the Aug. 14, 2003, northeastern U.S. blackout and caused the failure of an alarm system at First Energy's Akron, Ohio control center, slowing the response and allowing the blackout to cascade [8],
- Jaguar recalled 68,000 cars due to a software glitch that would automatically shift the car into reverse if it detected a significant change in fluid pressure [9].

Understanding the effects of software process improvements on these most severe error types is highly beneficial in making the case for investments in process

- *Donald Harter is with the Whitman School of Management, Syracuse University, Syracuse, NY 13244. E-mail: dharter@syr.edu.*
- *Chris F. Kemerer is with the Katz Graduate School of Business, University of Pittsburgh, Pittsburgh, PA 15260. E-mail: ckemerer@katz.pitt.edu.*
- *Sandra Slaughter is with the College of Management, Georgia Institute of Technology, Atlanta, GA 30308. E-mail: Sandra.slaughter@mgt.gatech.edu.*

Manuscript received (insert date of submission if desired). Please note that all acknowledgments should be placed at the end of the paper, before the bibliography.

improvement.

In addition to the severity of the defect, the *timing* of defect detection also affects software costs. A number of researchers have noted that it is significantly cheaper to fix defects detected earlier in the process, i.e., in development rather than in production [10], [11]. Development defects are those discovered prior to implementation of the system, while production defects are bugs found during systems operations after system implementation. Since production defects are believed to be the most expensive to repair, investments in process improvement early in the life cycle have the greatest potential to reduce costs.

Finally, not all *projects* are equal – some systems are large, others small, some complex, others less so. Some projects have requirements that are ambiguous, unclear or incomplete, or that may change frequently; other projects have requirements that are more straightforward and clearly defined. The software engineering literature has identified software complexity, size and requirements ambiguity as key factors influencing software quality [50]. Building on this we ask whether software process improvement has differential quality impacts for systems with different levels of these factors? To the degree that resources are limited where should improvements in software processes be prioritized: for larger systems, for more complex systems, or for systems with ambiguous requirements? An understanding of when software process improvement is most beneficial in terms of severe defect reduction can assist software managers in evaluating investment decisions by employing scarce resources where they provide the most benefit [12]. These research questions are summarized in Table 1.

<Table 1>

To answer these questions we analyzed an extensive dataset including detailed, longitudinal data collected over a period of twenty years on more than 7,545 development and production defects from a major software organization. The data were collected for an effort to develop a large software system at this organization. The software constitutes 7.7 million lines of code developed as part of an inventory management information system. The system development effort comprised a set of software projects which varied in size, complexity and ambiguity of requirements, with an average size of 175,000 lines of code and an average duration of 9.6 months. A unique feature of this setting is that it reflects a quasi-experimental design, in which the organization improved its software processes over time while working on various projects to develop the system, advancing from CMM level 1 through level 5 during the time period examined. By investigating projects conducted at different levels of CMM maturity for one system in one organization over time, the possible effects of many other extraneous factors are minimized. The longitudinal defect data thus constitute repeated measures of the quality of the processes used to develop this system, while controlling for the differences in software size, complexity and requirements ambiguity for each project, thus strengthening the internal validity of our findings. In addition, each defect was assigned a level of severity, allowing us to relate improvements in software development processes to defect severity. To the best of our knowledge this research is the first study that explores the effects of software process improvement as measured by the CMM level on the severity of software defects. Exploration of the interaction of CMM level with software size, complexity and requirements ambiguity also has the potential to lend more nuanced insights into what types of software projects are likely to experience the greatest benefits from improvements in software development processes.

In section 2 we review relevant prior literature on software quality and software process improvement. Section 3 proposes a research model and methodology to analyze the effects of software process improvement, software size, complexity, and requirements ambiguity on software defect severity. Section 4 describes the research site, research design and the unique data set. In section 5 we present the analyses and results. Section 6 discusses the implications and insights of the study for software engineering research and for managers in the software industry, and section 7 reviews the conclusions and opportunities for future research.

1 Previous Relevant Literature

2.1 Defect Severity

Although a significant volume of research has been published on software quality, much of the research has focused on cross-sectional studies and fault prediction models [13], [14], [15], [16], [17]. With few exceptions most of the research has not had access to data on defect severity, nor an opportunity to examine longitudinal data relating to an organization's evolution through all five CMM levels. In contrast to these fault prediction studies, the purpose of our current study is to empirically evaluate the effect of software process improvement, and software size, complexity and requirements ambiguity on defect severity.

Similar to prior researchers we define defects as flaws in the specification, design, or implementation of software [18] that cause the systems to fail to perform its function [19]. Defects can appear throughout the software life cycle [20]. Defect counts alone, while useful, may not provide a complete picture of the quality of software systems. A more thorough approach to categorizing errors is to examine the severity of a defect, as a system with even many cosmetic defects would be much more desirable to users than one with even a few catastrophic functional failures. Humphrey defines defect severity as the impact of a defect on the user's operational environment [21]. A number of software researchers specify a categorization of defects of up to four severity levels [18], [22], [23], [24]. The IEEE 729 standard defines the most important defects as either *Critical* (defect results in a systems or subsystem failure, usually crashing) or *Major* (functional failure of the system, e.g., incorrect calculations, data loss, etc.). A less important set of defects are either *Average* (system provides inconsistent or incomplete results) or *Minor* (defect does not cause failure or impair usability). This categorization is mirrored throughout the literature, with a slight variation in labeling, but relatively consistent inter-

pretations, by Jones [24], Poulin [26], Gao [27], UKSMA [28] and Tsui [29].

This prior literature on defect severity tends to be focused on standards or taxonomies, rather than on quantitative analysis of empirical severity data; such studies are rare, no doubt due to the reasons identified by Zhou and Leung that make severity data difficult to collect [30]. First, severity data is expensive and time-consuming to collect [30], [31]. Second, development or testing personnel must have sufficient training and background to accurately assess severity levels [32]. The difficulty in acquiring accurate severity data on software projects is likely the key reason that more empirical research has not been performed in this area.

Perhaps as a consequence of these difficulties, prior research has tended to aggregate the two top category defects into a single "functional/business-critical errors" category, and to aggregate the two bottom category defects into a "non-business-critical/inconveniences" category. Tsui further supports this two level view of defect severity by recommending that the top category defects should be fixed before production, but fixes for the bottom category defects can be deferred [29].

However, even with this two-level view there is very limited empirical research concerning the effect of process improvement on the severity of defects. Perhaps the most well known work is some anecdotal evidence from Hewlett-Packard (HP) suggesting that process improvements affect the frequency of high versus low severity defects. Their "10X" improvement program found a differential effect of process improvements on defect density for various severity levels [33]. In evaluating the effect of process improvement on defect severity HP aggregated critical and serious defects (category 1 and 2) as high severity defects and minor and cosmetic defects (category 3 and 4) as low severity defects. HP's five-year goals were to reduce post-release defect density by a factor of 10 (hence, the "10X" program) and to reduce the number of critical and serious defects by 10X. After nine years HP experienced a 6X reduction in defect density, but only a 2.5X reduction in critical and serious errors [33].

2.2 Software Process Improvement

For the last decade, software process improvement has been a primary approach to improving software quality [34]. Software process improvement refers to a defined framework of software procedures that define the steps and methods of a software process, define measures to assess and benchmark the process, and implement the defined procedures while looking for continuous improvement opportunities. Juran [35], [36], Deming [37] and Crosby [6] have long advocated process improvement as a means to improve quality in product development, manufacturing and services. Humphrey's early research in software engineering highlighted the need for process improvement in the software industry [38]. He drew on principles from manufacturing in developing quality oriented process guidelines for software development, which evolved into the Capability Maturity Model [39]. As the adoption and acceptance of the CMM model have spread, the research literature has become populated with numerous studies establishing a positive relationship between software process improvement and software quality [3], [40], [41], [42] and the relationships among process, quality and cost [4], [5], [43], [44], [45].

Our study extends this prior research by testing the relationship between software process improvement and defect severity, while controlling for other factors that may influence the results (the relationships are defined by and illustrated in our conceptual model in Figure 1 which is described in the next section).

3 Research Model, Design and Methodology

3.1 Conceptual Model

Prior research has tended to examine software quality using the project as the unit of analysis, most likely because the majority of organizations maintain quality statistics at this aggregate level. However, our study had access to uniquely detailed data about each defect. By moving the unit of analysis down to the defect level, we can explore the relationships between software size, complexity, requirements ambiguity and software process improvement on the severity level of each defect. Using these data we formulate a model with the defect as the unit of analysis and severity level as the dependent variable.

The conceptual framework for this analysis provides a model to estimate the probability of a detected defect being a high severity defect, given a set of conditions concerning software process improvement, software size, complexity, and requirements ambiguity. Intuitively, larger, more complex and more ambiguous systems are expected to experience worse problems, all else being equal, and we start by including those variables and testing them in our model. Then, with those controls included, we test for the effect of software process improvement on the potential for severe defects to occur.

Figure 1 outlines our two related models for (1) development defects (solid boxes and arrows) and (2) production defects (solid boxes and arrows plus dashed boxes and arrows). *Development defects* are those detected during the software development process but before customer acceptance. The strength of the relationship between software characteristics (size, complexity and ambiguity) and the severity of a defect is expected to be influenced by software process improvement. Process improvement is said to have a moderating effect on software characteristics and is depicted by the intersecting lines in Figure 1 [46]. A moderated model measures the effect that one variable has on the relationship between two other variables [47]. This interaction can strengthen or weaken the relationship between the system characteristics and the severe defects. In particular, in our study we use a moderated model to test whether the benefits of software process improvement are enhanced for systems that are larger, more complex or have more ambiguous requirements. From a practical perspective this can lend insight into what types of systems benefit the most from process improvements.

<Figure 1>

We propose that software process improvement, size, complexity and requirements ambiguity all directly affect

the likelihood that a defect will be classified as high severity. In Figure 1 these main effects are represented by the arrows from *Software* and *Process* to *High Severity Defects*. However, we argue that process improvement may have a differential impact on the probability of a high severity defect for software projects of different sizes, complexity levels, and levels of requirements ambiguity. These interactions are represented by the intersecting arrow in Figure 1.

Our research site maintained records for both development defects and production defects. The development defect model described above includes variables for process, size, complexity, ambiguity and their interactions. However, prior researchers have found that quality in one life cycle stage tends to influence quality in later stages [48], [49]. Because of this persistence of quality from development through production we include the *proportion* of high severity *development* defects as an additional variable in the production defect version of the model outlined in Figure 1 as represented by the dashed box and arrow. We separately examine models for development and production defect severity consistent with Figure 1 to determine whether the impacts on severity levels differ by life cycle phase.

3.2 Hypotheses

Using the models specified in Figure 1 we will examine both the direct effect of process improvement on the likelihood of development and production defects and whether process improvement interacts with the project level variables when determining the likelihood of high severity errors. The following sections specify our hypotheses for these effects.

3.2.1 Process Improvement Effect on Development Defect Severity

The CMM process maturity model encompasses processes to ensure quality throughout the development life cycle by monitoring the requirements, design, coding and testing process via functional requirement reviews, design reviews, code reviews and testing. One strength of this high level process is the ability to ensure that the final software matches the functional requirements from the start of the life cycle. For example, Jones found that severity level 1 errors were generally requirements errors, design errors dominated severity level 2, and coding errors and documentation errors were the primary source of errors for severity levels 3 and 4 [24]. The CMM focus on functional requirements and design would seem to be well-suited to reducing category 1 and 2 errors through the use of design walkthroughs, perhaps somewhat effective in reducing coding errors with code walkthroughs, but likely to be less focused on typographical and other cosmetic defects [41]. For this reason we propose that improvements in software processes will be more likely to reduce the likelihood of high severity defects in the development phase.

H1: *Software process improvement will decrease the likelihood of high severity development defects more than low severity development defects, all else being equal.*

3.2.2 Process Improvement Effect on Production Defect Severity

Earlier research has found that quality is persistent, *i.e.*, it tends to "cascade" through the software life cycle [48], [49]. This means that if quality is built into software early in the life cycle, then similar levels of quality will tend to manifest through later life cycle phases – e.g., higher quality designs lead to better code, fewer bugs in development testing, and fewer problems in production. All else being equal the quality of the *input* from one life cycle stage influences the quality of the *output* of subsequent stages. Building on this premise we predict that the benefits of process improvement during development will cascade to the production life cycle phase, reducing the likelihood of high severity production defects.

H2: *Software process improvement will decrease the likelihood of high severity production defects more than low severity production defects, all else being equal.*

3.2.3 Process Improvement, Complexity and Defect Severity

Research has found that software complexity has a significant effect on software development quality [50], [51], [52]. Highly complex software is more prone to have defects of all kinds, including the more severe defects [50]. Complexity also influences the software projects that are being enhanced in development [53] or maintained in production [54], [55]. And, it is possible that process improvement affects projects of varying levels of complexity in different ways. For example, the discipline of the CMM processes should ensure that complex projects successfully move through requirements definition, design, coding and testing. A less disciplined approach would potentially lose control of complex projects, resulting in a higher number and higher severity of defects. However, disciplined processes might accrue fewer benefits when building very simple systems, where the additional process structure is not required in order for developers to do a good job. Therefore, we propose that process improvement will moderate the relationship between complex software and high severity defects, such that:

H3: *Software process improvement will have a greater benefit for more complex software in reducing the likelihood of high severity defects, all else being equal.*

3.2.4 Process Improvement, Software Size and Defect Severity

A positive correlation between software size and the number of defects found in a software project is well-validated in the research literature [45], [48], [56], [57], [58], [59]. Software size influences the size of the software team, the level of integration necessary among components, and the increase in communication channels required among team members [60]. Larger projects thus impose what can be considered another dimension of complexity, *i.e.*, *management complexity*. Management complexity comes from the fact that larger software projects require increased team size, necessitating stronger internal team coordination processes. Increased software size also leads to decomposition into components with sub-teams, requiring component integration and inter-team coordination. Thus, given the additional activ-

ities required to integrate, coordinate and communicate in larger projects, it is quite likely that such projects would experience more defects, especially more severe defects. Indeed, the software development literature suggests that larger projects are significantly more prone to failure [57]. However, improved software development processes should be especially helpful for larger projects. Software process improvement includes the key processes for greater management discipline. Some examples of CMM key process areas which enhance management's ability to coordinate larger projects include configuration management, integrated project management, product integration, and project monitoring and control. Wang *et al.* reported that management controls lead to higher levels of project performance [61]. These disciplined management processes should improve the quality and reduce the severity of errors occurring in larger projects, hence:

H4: *Software process improvement will have a greater benefit for larger software projects in reducing the likelihood of high severity defects, all else being equal.*

3.2.5 Process Improvement, Requirements Ambiguity, and Defect Severity

The clarity of a functional requirement at the beginning of a software development effort has the potential to significantly determine if a software project is successful and thereby influence the resulting software quality [62]. The question is then how the effects of process improvement vary by projects with different levels of requirements clarity or lack thereof (ambiguity). One could argue that if a software project has very ambiguous requirements, then no process will guarantee a manager to successfully deliver high quality software, as high levels of ambiguity will result in misunderstandings of the requirements, misinterpretation, incorrect designs, and ultimately, more defects. Even a disciplined process will have difficulty interpreting a poorly written requirement. However, if a software requirement is clearly defined, the discipline of the CMM process could be expected to help to ensure that the software will be built to satisfy the requirement. Therefore:

H5: *Software process improvement will have a greater benefit for software with less ambiguous requirements in reducing the likelihood of high severity defects, all else being equal.*

4 Research Site, Design and Methodology

4.1 Research Setting

The analysis examines data collected on 6,190 development defects and 1,355 production defects in software developed over a twenty year period by the systems integration division of a large information technology corporation for a major customer. The software constitutes 7.7 million lines of code developed as part of an inventory management information system. During the time period of the development effort, the company implemented various software process improvement initiatives such as creation of life-cycle development standards, definition of detailed style guides for documentation, institutionalization of design review processes and program management status reviews, definition of schedule and performance metrics, use of Pareto analysis and the implementation of an automated cost estimation methodology, among others.

The research setting provides a quasi-controlled field environment to evaluate the effects of software process improvement on defect rates, and the severity of defects. During the development effort for this system the organization improved from CMM process maturity level one to level five. The defect data thus constitute repeated measures of the quality of this system at different levels of CMM process maturity, while controlling for differences in software size, complexity and requirements ambiguity.

4.2 Data Collection Methods

The software developer, the customer, and several independent outside organizations collected components of the detailed defect data used in this study. Independent assessors from other divisions of the software developer, customer personnel, and auditors used the SEI's CMM to evaluate the level of *software process improvement* to development and supporting activities.

Product measures including size, complexity, and requirements ambiguity were collected by the software developer and audited by the customer. The Configuration Management department at the software developer collected *software size* information using automated tools to ensure consistency in measurement. The customer confirmed software size through an independent count of lines of code. The Engineering department at the software developer determined *ex ante* complexity and requirements ambiguity ratings based on the customer's functional requirement specification using the criteria identified by Jones [24]. Prior to the start of any software development, the customer performed an independent analysis of complexity and requirements ambiguity. If there were any discrepancies between the software developer's analysis and the customer's analysis, the two parties met prior to the start of a project to review and resolve any differences.

Defects were discovered in testing either prior to customer acceptance (*development defects*), or after a system was implemented in production (*production defects*). Testing during the development phase was first performed by the software developer, then by the customer in their acceptance testing. The software developer's test department used a customer-approved test plan to test the software against the design specification. The description and severity of errors found in this testing were recorded by the developer's test team. The customer performed a functional systems test of each software project with a similar plan, first using a sample database for functional testing, and then using a full-scale database for stress testing. All of the defects found by the developer or customer during these development and acceptance tests were included in the development defect database. After system acceptance by the customer all further defects were identified as production defects. Users in the customer firm identified errors in production and recorded the description and severity of these production defects.

The customer audited defects identified by the developer; likewise, the developer reviewed and authenticated errors identified by the customer. Assignment of severity

levels received similar scrutiny by both parties and followed a documented procedure to ensure consistency of assignment over time. Similar to how Hewlett-Packard implemented its Defect Tracking and Software Metrics Data Base, the configuration management department at the development firm maintained all software problem reports in an electronic database [32]. The data examined in this study were extracted from these electronic files and some additional hardcopy files maintained by the engineering and configuration management departments at the software development firm.

4.3 Construct Measurement

As noted earlier, in prior studies that have analyzed empirical defect data by severity, the researchers have aggregated their data into two levels, e.g., HP's Grady aggregates his severity levels into two levels: major and minor [33]. Major defects are those which result in a functional failure of the system (a category 1 "crash" or category 2 "functional failure"). Minor defects may create an inconvenience (category 3 "minor errors" or category 4 "cosmetic errors"), but do not interfere with business-critical activities [63]. Jones also found this grouping in empirical data, where category 1 and 2 defects were often resolved before software acceptance, but category 3 and 4 defects were deferred [24]. He found that category 1 and 2 defects constituted 3% and 47% of his sample[1].

Similarly, in our current work we believe that high and low severity are determined by whether a defect is business critical or non-business critical. Therefore, we have followed the approach taken by other researchers and aggregated the four severity levels into two, labeling severity categories 1 and 2 as "high severity", and categories 3 and 4 as "low severity" [24]. Using this categorization of business-critical and non-business critical errors, *high-severity development* defects are identified prior to customer acceptance and categorized as critical or serious. *Low-severity development* defects are defects that are identified in development prior to customer acceptance, but categorized as minor or cosmetic errors. Similarly, *high-severity production* defects are identified by users in a production environment and categorized as critical or serious. *Low-severity production* defects are identified by users in production and categorized as minor or cosmetic errors. For production defects the software developer was contractually required to fix high severity defects quickly, while low severity defects could be deferred to subsequent version releases. After submission of a production defect to the defect tracking system, the developer, customer, and users reviewed each defect and its classification to ensure that the severity assigned conformed to the standards established within the contract. This process also ensured that submissions were defects and not requests for new software features or enhancements.

Table 2 summarizes the description of the key measures of variables in our study.

<Table 2>

From Table 2 we have six software measures. As noted above, *High severity defects* are defined as business critical functional failures or program termination. *Low severity defects* are defined as non-business critical minor disruptions and cosmetic errors [24]. Of the 6,190 total development defects, 57.89% were high severity (averaging 50 high severity development defects per product). Similarly, of the 1,355 total production defects, 61.1% were high severity (averaging 11 high severity production defects per product).

Software process improvement is the level of discipline and process control in the software development process [39]. We measure process improvement using the SEI CMM level of maturity. Each software project has a maturity level defined at the start of design adjusted commensurately for process improvements during the development life-cycle of that software. The CMM evaluation process involves the examination of Key Process Areas (KPA) relevant to each of the five CMM levels. Advancement from one CMM level to the next requires adherence to a preponderance of the KPAs for the level. This process improvement measurement approach has been employed successfully in prior research [3, 4, 5]. CMM level 1 implies that processes are ad hoc; processes might exist but are not used uniformly through the organization. At CMM level 2, processes are repeatable; processes are used consistently throughout the organization, but there is limited training and documented standards. CMM level 3 includes defined standards and mandatory training. CMM levels 4 and 5 introduce measurement and continuous process improvement respectively. The CMM process maturity model provides a consistent measure of process improvement that can be used in longitudinal studies. The CMM was replaced by the CMMI in 2002 and post-2002 process maturity levels were measured using the updated approach[2]. The CMM maturity level of a software project is pro-rated based on the fraction of development performed at each level [49], [64].[3] In the first twelve years of the effort CMM assessments were performed approximately every two years, and the company's CMM maturity level increased from level 1 to level 3 during that period. In the thirteenth year the CMM level fell to level 2, but then returned to level 3 in year fourteen, and in the final years of data collection was assessed at level 5. All CMM and CMMI appraisals were performed by teams external to the research site.

Software size is a measure of the magnitude of the software in thousands of lines of code (KLOC). LOC has a long tradition of use as a measure for size in software research [41], [45], [56], [59]. Although LOC has sometimes been challenged as a potentially inconsistent meas-

[1] The lone exception to this approach is the work of Zhou and Leung who chose to separate major and minor defects by declaring only category 1 defects as major; minor defects included categories 2, 3, and 4. This may be because in their dataset of 549 defects, only 34 were category 1 defects (6% of the sample), but category 2 defects comprised 77% of their data. Combining these would mean that 83% of their sample would have been considered "high severity". By contrast, in our data about 60% are categories 1 and 2 and 40% categories 3 and 4, a distribution similar to Jones's.

[2] For more on CMMI (Capability Maturity Model Integration), see *CMMI Distilled (2nd Ed.): A Practical Introduction to Integrated Process Improvement* by Dennis M. Ahern, Aaron Clouse, and Richard Turner; Addison-Wesley 2004, 305 pages.

[3] For example, a software product which spanned twelve months, six months under CMM level 2 practices and six months under CMM level 3 practices, would have a weighted process maturity score of 2.5.

ure, the use of a single programming language at our research site and the automated counting of lines of code make its use appropriate in this context.

Software complexity reflects three dimensions of complexity: domain, data and decision complexity [5], [24]. *Domain complexity* measures functional and algorithmic sophistication (Table 3). *Data complexity* quantifies the number of data variables and the complexity of data relationships (Table 4). *Decision complexity* reflects the structural and procedural complexity within a program (Table 5). The three complexity measures are based on techniques developed by Jones and implemented in a number of software cost, schedule and quality models [24]. The measures use a Likert scale from 1 to 5 as described in Tables 3, 4, and 5, with higher values indicating higher complexity. Since the three measures tend to be highly correlated, *software complexity* is the calculated as the average of these three measures [5].

<Table 3>

<Table 4>

<Table 5>

Requirements ambiguity is an assessment of the clarity and specificity of the users' requirements as documented in a functional specification. Requirements engineering and clarification is a crucial step in the software engineering life cycle [62]. Requirements ambiguity is measured on a scale of 1 to 5 (Table 6), with higher values indicating greater ambiguity [24].

<Table 6>

4.4 Data Validity

A particular and unique strength of the data collected for this study concerns the level of rigor with which the data were measured. All data collected for this study were audited for correctness and completeness by various individuals and organizations. A summary of audit reviews and reviewers is shown in Table 7.

<Table 7>

4.5 Data Collection Summary

External executives from other divisions of the company and client auditors ensured the accuracy of *ex ante* software process improvement ratings. The chief engineer and client technical reviewers audited *ex ante* product characteristics, *i.e.*, complexity and requirements ambiguity. Quality assurance and client personnel reviewed software size and quality measurements.

Descriptive statistics for development and production defect data are displayed in Tables 8 and Table 9.

<Table 8>

<Table 9>

Tables 10 and 11 present the correlation matrices for the development and production variables. As might be expected, higher design complexity is associated with greater product size. Products with higher complexity also appear to be more ambiguously defined, also as might be expected. Perhaps less intuitive, however, is the negative correlation between size and requirements ambiguity. However, in reviewing the data from the research site, it turned out that online updates and batch reports tended to be the larger programs. As the customer tended to provide clearer requirements for update and report software, this is manifested as a negative correlation in the data.

<Table 10>

<Table 11>

5 Data Analysis and Results

The model in Figure 1 identified the relationships among process improvement, size, complexity, and requirements ambiguity and the likelihood of a high severity defect occurring. We earlier categorized defects as either high or low severity and our model has a binary dependent variable where severity is coded as a 1 for high severity and 0 for low severity.

Ordinary least squares regression is not appropriate when the dependent variable is categorical as is the case here. Since we are interested in the likelihood of encountering a high severity error when a defect is detected, it is appropriate to use probit or logit to estimate the model coefficients. Probit and logit models often find similar results, although probit is more sensitive to values near the mean and logit is more sensitive to extreme values [65]. Greene [66] suggests that there are no clear criteria for the selection of probit versus logit, and therefore we estimate a probit model, with a logit analysis as a robustness check on our results.

In each of the equations below the dependent variable is the binary variable identifying if a defect is classified as high severity or low severity. This model will allow us to predict the probability of encountering a high severity defect. The intercept term in each equation reflects the probability of a high severity error when size, complexity, ambiguity and process improvement are at their respective mean values. Note that, for ease of interpretation, we followed common practice and centered the independent variables in our analysis, *i.e.*, we subtracted each value for a variable from its respective mean value[4]. The error term is represented by epsilon.

$$P(\text{Development high-severity}=1) = \beta_{01} + \beta_{11}*Size + \beta_{21}*Complexity + \beta_{31}*Ambiguity + \beta_{41}*CMM + \beta_{51}*CMM*Size + \beta_{61}*CMM*Complexity + \beta_{71}*CMM*Ambiguity + \varepsilon_1$$

$$P(\text{Production high-severity}=1) = \beta_{02} + \beta_{12}*\text{Percent High Severity Development Defects} + \beta_{22}*Size + \beta_{32}*Complexity + \beta_{42}*Ambiguity + \beta_{52}*CMM + \beta_{62}*CMM*Size + \beta_{72}*CMM*Complexity + \beta_{82}*Ambiguity + \varepsilon_2$$

Probit regression uses the normal distribution to estimate the coefficients of the equation:

$$P(\text{Development high-severity}=1) = \Phi(\Sigma\beta_{i1}X_i)$$

$$P(\text{Production high-severity}=1) = \Phi(\Sigma\beta_{i2}X_i)$$

Prior research has shown that defect rates decrease as

[4] For further details on this statistical procedure see Aiken, L. S., & West, S. G. (1991). *Multiple Regression: Testing and interpreting interactions*. Newbury Park, CA: Sage.

process improvement increases [4], [5]. The probit analyses will project whether the proportion of high-severity errors or low-severity errors is expected to decrease more quickly. If the coefficient on process improvement is positive, then the proportion of high-severity errors is predicted to increase with improved processes, *i.e.*, low-severity errors are declining faster. However, if the coefficient is negative, then increases in process improvement are associated with faster declines in high-severity errors. Since we predict that process improvement will have a greater impact on high-severity errors and a lesser impact on low-severity errors, for our hypothesis to hold, the coefficient on process-improvement is expected to be negative. Table 12 shows the parameter estimates for the probit analysis of development errors. A similar analysis is performed for production errors and is shown in Table 13. The models were examined hierarchically, first with controls only, then controls and CMM, and finally including interactions

<Table 12>

<Table 13>

The results for the probit analyses of both the development and production defect severity indicate that each of the models is statistically significant (see Wald statistics in Table 12 and Table 13). The following sections address each of the hypotheses.

5.1 Main Effect: Process Improvement and Defect Severity

Hypothesis 1 predicted that increases in process improvement would reduce the likelihood of development errors. The process improvement coefficient in Table 12 is negative and significant at the α =.05 level (β_{41}=-0.069, p=0.03), suggesting support for the hypothesis.[5] Marginal effects analysis of the process improvement coefficient supports this conclusion (dF/dx= -0.027, p=0.03). Earlier studies have shown that overall defect rates decrease with improvements in process improvement [5]. These results further indicate that as the organization improved its processes, the likelihood of a defect being a high severity defect also decreased.

Hypothesis 2 similarly predicted that process improvements would lead to reduced likelihood of high severity production defects due to the persistence of quality throughout the life cycle [48]. The coefficient for process improvement in the production equation in Table 13 is also negative and significant (β_{52}=-0.370, p<0.01). Marginal effects analysis of the process improvement coefficient supports this conclusion (dF/dx= -0.140, p<0.01). Hypothesis 2 is supported; the quality effects of process improvement improvements cascade from development through the production phase of the software development life cycle. As hypothesized, building quality into the product in early life cycle stages appears to influence the quality at later stages.

Figure 2 graphically illustrates how process improvements relate to defect severity in software development and production over the time period of the development effort at our research site. As can be seen in Figure 2, levels of process improvement, as measured by CMM levels, are inversely related to the likelihood of high severity development defects and high severity production defects. When the CMM level is higher, the proportion of high severity development and production defects is lower, and when the CMM level is lower, the proportion of high severity development and production defects is higher.

<Figure 2>

5.2 Interaction Effect: Process Improvement and Complexity

Hypothesis 3 proposed that process improvement moderated the effect of complexity on the probability of encountering a high severity defect. Although we would intuitively expect that increases in complexity might increase the likelihood of high severity errors, we proposed that process improvements would reduce the likelihood for highly complex systems due to the disciplined processes involved at higher levels of process improvement. A test of the interaction effect proposed by Ai and Norton [67] and Norton *et al*. [68] for probit models confirms that the interaction term is significant for both development and production (one-sided z_D=-3.752, p<0.01; z_P=-3.478, p<0.01).

In order to provide a stronger understanding of the interaction effects, the results are also illustrated graphically in Figure 3. The lines in the figure represent the proportion of high severity development defects at different levels of complexity for CMM levels 1 through 5. One line (the dashed line) is graphed at the mean level of complexity, another line (the dotted line) is graphed at a low level of complexity (mean – one-half standard deviation), and the third line (the solid line) is graphed at a high level of complexity (mean + one-half standard deviation).

<Figure 3>

As can be seen in Figure 3 at low levels of complexity the proportion of high severity development defects does not differ significantly at CMM level 1 (61.33%) versus at the highest levels of CMM process maturity (60.19% at CMM level 5). This is shown graphically in Figure 3 as the dotted line for low complexity is essentially flat from CMM level 1 to CMM level 5. However, at high levels of complexity, the proportion of high severity development defects is higher when the organization is operating at CMM level 1 (65.84%) than at CMM level 5 (45.38%), as seen in Figure 3. The downward sloping line for high levels of complexity from CMM level 1 to CMM level 5 suggests that, at high levels of complexity, there are in-

[5] In a model with interaction effects it is important to evaluate main effects in the complete model. Otherwise the evaluation would suffer from an omitted variables bias [66]. Mathematically, when one evaluates the main effect of a variable in the complete model (including interactions), one must differentiate the model with respect to that particular variable in order to obtain the relevant coefficients and standard errors for the main effect. For example, differentiating the Development Defects model with respect to the CMM variable yields: β_{41} + β_{51}*size + β_{61}*complexity + β_{71}*ambiguity. By convention one substitutes the mean values of size, complexity and ambiguity to determine the overall coefficient on the CMM. Since we have centered our independent variables, centered size, complexity and ambiguity all have means of 0. Given that, the coefficients on β_{51}, β_{61} and β_{71} will drop out; this leaves β_{41} (with a value of -0.069) for the main effect of the CMM in the development model. To interpret the coefficient on β_{41} we need to estimate the change in the probability of a severe defect occurring given an "infinitesimal" change in the CMM variable. This is the marginal effect (dF/dx) and is estimated using the DPROBIT command in the Stata statistical software. We use a similar process to evaluate the main effect of the CMM in the production model.

creasing benefits from higher levels of process improvement in terms of a reduced likelihood of severe development defects. This implies that the disciplined approaches of CMM have a magnified benefit for highly complex projects, but make less difference for simple projects. Therefore, Hypothesis 3 is supported for development defects.

This pattern is also apparent for production defects in Figure 4. As in development, higher levels of process improvement reduce the likelihood of high severity defects in production even more for more complex projects. The cascading effect of quality, or the persistence of quality, is magnified at later life cycle stages. Similar to development, higher levels of process improvement have a greater impact on more complex systems in reducing the probability of high severity production defects. Hypothesis 3 is thus supported for production defects.

<Figure 4>

5.3 Interaction Effect: Process Improvement and Size

In Hypothesis 4 we posited that the size of a project would have a similar effect on high severity defects as complexity. Complexity measures the internal complexity of the product design, but the size of the software developed imposes management complexity on a project. We proposed that process improvement would have a more significant effect in reducing high severity errors for larger projects, since a more disciplined management approach would be well-suited to large scale software projects. The probit model interaction test of process improvement and size for development defects finds the coefficient negative and significant (one-sided z_D=-4.595, $p<0.01$) [67], [68].

Figure 5 graphs the interaction between project size, software process improvement and development defect severity. As shown in Figure 5 there is little difference in the proportion of high severity development defects for small projects at CMM Level 1 versus small projects at CMM Level 5. However, for large projects, the proportion of high severity development defects is significantly lower at CMM level 5 (44.26%) than at CMM Level 1 (68.85%). Thus, higher levels of process improvement appear to have a greater benefit for larger projects than smaller ones, supporting Hypothesis 4.

<Figure 5>

The pattern is similar for production defects. The probit interaction test confirms that the interaction is negative and significant (one-sided z_P=-1.739, $p<0.05$) [67], [68]. The dominance of process improvement over any interaction with size is apparent in Figure 6, where higher levels of process improvement significantly reduce the likelihood of high severity production errors for all size levels. The downward sloping line (lower, solid line) indicates that higher levels of process improvement are especially beneficial for large software projects in terms of reducing the likelihood of severe production defects. Therefore, Hypothesis 4 is also supported for production defects.

<Figure 6>

5.4 Interaction Effect: Process Improvement and Requirements Ambiguity

Hypothesis 5 predicted that process improvement would have a greater benefit for clearly defined software products. We reasoned that when requirements ambiguity is high the selection of a development process will make little difference since the goal for the project is not clearly defined. The coefficient for the main effect of process improvement on development severity was marginally negative, as discussed before. However, the interaction test by Norton *et al*. finds that the interaction coefficient is positive and significant (z_D=3.039, $p<0.01$) [67], [68]. The sign of this coefficient means that when the requirement is clearly defined, process improvement has the benefit of reducing high severity likelihood. At higher levels of requirements ambiguity there are offsetting effects (a reduction in severity due to the main effect of process improvement, but an increase in severity due to the interaction with ambiguity), which indicate that process improvement has a limited effect at higher levels of requirements ambiguity. This is evident from Figure 7 - at low levels of requirements ambiguity there is much more benefit from improvements in process maturity, in terms of a reduction in the proportion of high severity development defects, than at high levels of requirements ambiguity. As indicated in Figure 7, at low levels of requirements ambiguity, the proportion of high severity development defects is 64% at CMM level 1, but drops to 46.62% at CMM level 5. However, at high levels of requirements ambiguity there is little difference in the proportion of high severity development defects at CMM level 1 versus at CMM level 5. This statistical result is reminiscent of the management adage, "If you don't know where you're going, any road will get you there". Hypothesis 5 is thus supported for high severity development defects.

<Figure 7>

We find a somewhat different pattern for production defects. The main effect of process improvement is negative and significant as previously discussed, but the coefficient for the interaction of process improvement and ambiguity is not significant at usual levels using the probit interaction test (one-sided z_P=0.359, $p=0.36$) [67], [68]. This is reflected in the lines in Figure 8 which are all downward sloping and are almost parallel, indicating a main effect due to process improvement, but little or no interaction with ambiguity. Thus, Hypothesis 5 is not statistically supported for production defects at the same level that it was supported for development defects. This pattern may be evidence that the cascading or persistent effect of quality overwhelms the interaction effect with requirements ambiguity for later software development life cycle stages.

<Figure 8>

6 Discussion

6.1 Summary of Results

The results of our hypotheses for both the development and production models are summarized in Table 14 below.

<Table 14>

6.2 Discussion of results

The results of this study lend insight into how to gain the greatest benefit from CMM process improvements in terms of reducing the likelihood of severe errors. Overall, simply advancing in levels of process improvement reduces the likelihood of high severity defects and therefore decreases the incidence of catastrophic or major functional failure. The greater effect on production vs. development defects highlights the importance of process improvement to the operational environment of production systems. Production defects are the most costly and their removal can have significant financial benefit to users and day-to-day operations [1].

We further found that higher levels of process improvement have the greatest leverage (in terms of reducing severe defect occurrence) on large projects, complex projects, and clearly defined projects. The management discipline inherent in higher levels of improvement appears to solve management complexity issues of coordinating large scale projects. It is likely that the strict design review processes, configuration management control and quality assurance review ensure that sizeable projects avoid coordination and integration issues that typically plague larger software projects. The practice of reducing software development risk by breaking large projects into component projects has often been advocated as a solution to managing large projects. Software process improvement is an alternative management approach that can improve the quality of large projects and reduce defect severity.

Technical complexity has also been a risk factor in managing software development. As expected, we find that complexity increases the likelihood of high severity defects. However, process improvement appears to be a very effective means for countering the risks of increasingly complex systems. Since complexity is often inherent in the application and therefore not under the direct control of the software project manager, applying process improvement may be the most effective recourse the manager has.

The ambiguity of a customer's requirements can clearly affect the ultimate quality of the product, as seen in our results. Since process improvement has a greater impact when the requirement is clearly defined, managers should ensure that sufficient efforts are undertaken to clarify requirements when needed. Investments in reviewing ambiguous requirements with a client, clarifying issues and features, can have a significant effect on the quality of the system. Our findings suggest that investments in requirements clarification and process improvement are not substitutes for each other, but instead tend to be complementary.

The payoff of process improvement is the reduction of high severity defects which result in operational failures. These defects are more costly to repair, result in organizational downtime, and have greater impact on the users. The reduction in these expensive defects is more dramatic for larger and more complex products, which inherently have more risk. Executives can use CMM-based process improvement to simultaneously manage development risk while reducing the impact to the organization.

6.3 Sensitivity Analysis, Strengths and Limitations

As in any empirical study there are potential alternative factors that could influence the likelihood of high severity defects. Team experience, team turnover, team size, number of software release versions, code reuse and software language all have the potential to affect the frequency of high severity defects. A sensitivity analysis found that team experience was not statistically significant in either the development model ($p=0.298$) or the production model ($p=0.163$). Team turnover was also not significant in either model (development $p=0.284$; production $p=0.318$). Similarly, neither team size (development $p=0.139$; production $p=0.476$) nor number of software release versions was significant (development $p=0.810$; production $p=0.125$). With respect to potential code reuse all data used during the estimation process of each of the software products were available from the corporate archives. In the estimation model input parameters all teams had set the percent of code reuse to zero. Subsequent conversations with management at the research site confirmed that there was no code reuse during the period of the study. Management also confirmed that the same programming language was used for all of the projects on the system.

Of course, the research reported in this paper has certain strengths as well as limitations, as do all such empirical studies. The major limitation of this study is that it relies on data from a single organization, and may not generalize to others. However, the use of a single firm allowed us to control for factors that might have skewed the results, such as development methodology, hiring policies, or management structure, as each of these factors were consistent throughout the life of the study, but might vary if data from multiple organizations had been combined. By focusing on a single organization and studying its processes and data in depth we approximate a quasi-experimental study where variables not of interest are held constant. Overall, given the nature of the software processes studied we believe that the broad results represent sound underlying principles of process improvement and are likely to be able to be generalized across other software development organizations.

The costs of defects are often difficult to quantify when the operational impacts are included. Operational downtime, lost organizational productivity, and financial impact to the day-to-day operations of a company should be included for an exhaustive list of costs per defect. Since only high severity defects have operational impacts, it is clear that the return on investment for preventing high severity defects is understated. Better quantification of the cost prevention benefits would allow managers to make more informed decisions when evaluating software process improvements.

7 Conclusions

This research addresses an issue of practical importance, i.e., the relationship between software process improvement and the severity of defects in the produced software-based systems. A significant amount of data that stems from many projects performed during a long time

span was analyzed in order to better understand this relationship and derive guidance for process improvement. Specifically, we examined an extensive dataset over a twenty year period from a CMM level 5 organization including software projects averaging 175,000 lines of code with varying levels of process improvement, size, complexity and requirements ambiguity. Over 7,500 defects were recorded and analyzed. Having data collected over such a long period is an unusual opportunity, and analyzing this data has important implications for process improvement. This unique research opportunity allowed us to evaluate the effect of each of key design parameters on defect severity during software development and later in production and operations. The results are supportive of the hypotheses in our model and lend insight into when process improvements are most beneficial.

In general, our findings indicate that process improvement reduces defect severity for both the development and production life cycle phases. The effect of process improvement on defect severity in production tends to be stronger than the effect in development, and we suggest that this might be evidence of the persistence or cascading effect of quality throughout the software development life cycle.

Our results also reveal that process improvement has differential effects on reducing defect severity for each of our design variables of interest. Although process improvement reduces the likelihood of defects for all levels of complexity, it has a greater effect on the reduction of high severity defects for more complex products. We find that the highly structured, disciplined processes of the CMM have a greater payoff when a software system is above average in complexity.

The effect of the CMM on severity for larger projects mirrors the results found for complexity. Increasing the size of a project introduces management complexity. We find that higher levels of process improvement have greater benefits in reducing defect severity for large scale projects. The disciplined management processes at higher levels of the CMM provide the ability to manage projects with more integration and staff coordination requirements.

In contrast, higher levels of the CMM are less effective in reducing defect severity when requirements are ill defined. Requirements ambiguity dampens the benefits of process improvement when considering the severity of errors. As we have noted, disciplined processes are not as useful when it is not clear what the software should do. Instead, for clearly defined software products higher levels of CMM processes appear to effectively reduce defect severity.

Our findings provide a detailed analysis of the benefits and limitations of software process improvement for the removal of severe development and production defects and suggest where investments in improving development processes may have their greatest effects.

REFERENCES

[1] *The Economic Impacts of Inadequate Infrastructure for Software Testing*, U.S. Department of Commerce, National Institute of Standards and Technology, 2002

[2] C. Mann, "Why software is so bad … and what's being done to fix it," MIT Technology Review, 2002

[3] J. Herbsleb, D. Zubrow, D. Goldenson, W. Hayes, M. Paulk, "Software quality and the Capability Maturity Model," *Comm. of ACM*, vol. 40, no. 6, 1997

[4] M.S. Krishnan, C.H. Kriebel, S. Kekre, T. Mukhopadhyay, "An Empirical Analysis of Productivity and Quality in Software Products," *Management Science*, vol. 46, no. 6, 2000.

[5] D.E. Harter, M.S. Krishnan, and S.A. Slaughter, "Effects of Process Maturity on Quality, Cycle Time and Effort in Software Product Development," *Management Science*, vol. 46, pp. 451-466, 2000.

[6] P.B. Crosby, *Quality is Free*, McGraw-Hill, 1979.

[7] C. Jones, "Measuring Defect Potentials and Defect Removal Efficiency,"*CrossTalk*, June2008, http://www.stsc.hill.af.mil/crosstalk/2008/06/0806Jones.html

[8] K. Belson and M.L. Wald, "'03 Blackout is Recalled, Amid Lessons Learned," New York Times, August 13, 2008.

[9] G. Prophet, "A hard look at software reliability," Global EDN Report, June23,2005. http://www.edn.com/article/CA608882.html?industryid=23439

[10] B.W. Boehm, *Software Engineering Economics*, Prentice Hall, 1981

[11] M. Lipow, "Prediction of Software Failures," *The Journal of Systems and Software*, vol. 1, no. 1; pg. 71, 1979.

[12] J.A. Hager, "Software Cost Reduction Methods in Practice," *IEEE Trans. Software Eng.*, vol. 15, no. 12; pp. 1638-1644, 1989.

[13] T. Gyimothy, R. Ference, and L. Siket, "Empirical Validation of Object-Oriented Metrics on Open Source Software for Fault Predictions," *IEEE Trans. Software Eng.*, vol. 31, no. 10, pp. 897-910, 2005.

[14] M. Alshayeb and L. Wei, "An Empirical Validation of Object-Oriented Metrics in Two Different Iterative Software Processes," *IEEE Trans. Software Eng.*, vol. 29, no. 11, pp. 1043-1049, 2003.

[15] K. El Emam, S. Benlarbi, N. Goel, and S.N. Raj, "The Confounding Effect of Class Size on the Validity of Object-Oriented Metrics," *IEEE Trans. Software Eng.*, vol. 27, pp. 630-650, 2001.

[16] J. Lo, C. Huang, "An integration of fault detection and correction processes in software reliability analysis," *The Journal of Systems and Software*, vol. 79, no. 9; pg. 1312, 2006.

[17] K.-Y. Cai, "On estimating the number of defects remaining in software," *The Journal of Systems and Software*, vol. 40, no. 2; pp. 93-114, 1998.

[18] R.B. Grady, *Practical Software Metrics for Project Management and Process Improvement,* Prentice Hall, Englewood Cliffs, New Jersey, 1992.

[19] S. Zahran, *Software Process Improvement: Practical Guidelines for Business Success*, Essex, England: Addison Wesley Longman Ltd., 1998.

[20] R. Chillarege, Orthogonal Defect Classification, *Handbook of Software Reliability Engineering*, IEEE Computer Society Press, pp. 359-400, 1995.

[21] W.S. Humphrey, *Managing the Software Process*, Addison-Wesley Publishing, Reading, Massachusetts, 1989.

[22] M. Donnelly, B. Everett, J. Musa and G. Wilson, Best Current Practice of Software Reliability Engineering (SRE), *Handbook of Software Reliability Engineering*, IEEE Computer Society Press, pp. 219-254, 1995.

[23] L.H. Putnam and W. Myers, *Industrial Strength Software: Effective management Using Measurement*, IEEE Computer Society Press, pp. 44, 1997.

[24] C. Jones, *Applied Software Measurement: Assuring Productivity and Quality*, McGraw-Hill, New York, 1996.

[25] ANSI/IEEE Std 729-1983 *Glossary of Software Engineering Terminology*, Institute of Electrical and Electronics Engineers. New York; IEEE. 1987.

[26] J.S. Poulin, *Measuring Software Reuse, Principles, Practices, and Economic Models*, Addison-Wesley, 1997.

[27] J. Gao, H.J. Tsao, Y. Wu, *Testing and Quality Assurance for Component-based Software*, 2003.

[28] *Quality Standards Defect Measurement Manual*, United Kingdom Software Metrics Association (UKSMA), October 2000.

[29] F. Tsui, *Managing Software Projects*, Jones & Bartlett Publishing, 2004.

[30] Y. Zhou and H. Leung, "Empirical Analysis of Object-Oriented Design Metrics for Predicting High and Low Severity Faults," *IEEE Trans. Software Eng.*, vol. 32, no. 10, pp. 771-789, 2006.

[31] T.J. Ostrand and E.J. Weyuker, "The Distribution of Faults in a Large Industrial Software System," *Proc. Int'l Symp. Software Testing and Analysis*, pp. 55-64, 2002.

[32] T.J. Ostrand, E.J. Weyuker, R.M. Bell, "Predicting the Location and Number of Faults in Large Software Systems, *IEEE Trans. Software Eng.*, vol. 31, no. 4, pp. 340-355, 2005.

[33] R.B. Grady, *Successful Software Process Improvement*, Prentice Hall, Englewood Cliffs, New Jersey, 1997.

[34] T. Dyba, "An Empirical Investigation of the Key Factors for Success in Software Process Improvement," *IEEE Trans. Software Eng.*, vol. 31, no. 5, pp. 410-424, 2005.

[35] J.M. Juran, "A Note on Economics of Quality," *Industrial Quality Control*, pp. 20-23, 1959.

[36] J.M. Juran, *Juran on Quality by Design: The New Steps for Planning Quality into Goods and Services*, Free Press, 1992.

[37] W.E. Deming, *Out of Crisis*, MIT Center for Advanced Engineering Study, 1992.

[38] W.S. Humphrey, "Characterizing the Software Process: A Maturity Framework," *IEEE Software*, vol. 5, no. 3, pp. 73-79, 1988.

[39] M.C. Paulk, B. Curtis, M.B. Chrissis, and C.V. Weber. "Capability Maturity Model, Version 1.1," *IEEE Software*, vol. 10, no. 4, pp, 18-27, 1993.

[40] G. Li and S. Rajagopalan, "Process Improvement, Quality and Learning Effects," *Management Science*, vol. 44, pp. 1517-1532, 1998.

[41] M.S. Krishnan and M.I. Kellner, "Measuring Process Consistency: Implications for Reducing Software Defects," *Management Science*, vol. 25, pp. 800-815, 1999.

[42] N. Ramasubbu, S. Mithas, M. S. Krishnan, and C. F. Kemerer "Work Dispersion, Process-Based Learning and Offshore Software Development Performance", *MIS Quarterly*, v. 32, n. 2, pp. 437-458, June 2008.

[43] H. Wohlwend and S. Rosenbaum, "Schlumberger's Software Improvement Program," *IEEE Trans. Software Eng.*, vol. 20, no. 11, pp. 833-839, 1994.

[44] M. Diaz and J. Sligo, "How Software Process Improvement Helped Motorola," *IEEE Software*, vol. 14, no. 5, pp. 75-81, 1997.

[45] M. Agrawal and K. Chari, "Software Effort, Quality and Cycle

Time: A Study of CMM Level 5 Projects," *IEEE Trans. Software Eng.*, vol. 33, no. 3, pp. 145-156, 2007.

[46] M.C. Paulk, C.V. Weber, B. Curtis, M.B. Chrissis, *The Capability Maturity Model: Guidelines for Improving the Software Process*, Addison-Wesley Publishing Company, Reading, MA, 1994.

[47] E. Barry, C. F. Kemerer, and S. Slaughter "How Software Process Automation Affects Software Evolution: A Longitudinal Empirical Analysis", *Journal of Software Maintenance and Evolution*, v. 19, n. 1, pp. 1-31, January-February 2007.

[48] C. Andersson, P. Runeson, "A Replicated Quantitative Analysis of Fault Distributions in Complex Software Systems," *IEEE Trans. Software Eng.*, vol. 33, no. 5, pp. 273-286, 2007.

[49] D.E. Harter, S.A. Slaughter, "The Cascading Effect of Process Maturity on Software Quality," *ICIS Proceedings*, 2000.

[50] C.F. Kemerer, "Software Complexity and Software Maintenance: A Survey of Empirical Research", *Annals of Software Engineering*, v. 1, n. 1, pp. 1-22, August 1995.

[51] J.W. Cangussu, R.A. DeCarlo, and A.P. Mathur, "A Formal Model of the Software Test Process," *IEEE Trans. Software Eng.*, vol. 28, no. 8, pp. 782-796, 2002.

[52] B.K. Clark, "Quantifying the Effects of Process Improvement on Effort," *IEEE Software*, vol. 17, pp.65-70, 2000.

[53] R.D. Banker and S.A. Slaughter, "The Moderating Effects of Structure on Volatility and Complexity in Software Enhancement," *Information Systems Research*, vol. 11, no. 3, pp. 219-240, 2000.

[54] R.D. Banker, S.M. Datar, C.F. Kemerer, and D. Zweig, "Software Complexity and Software Maintenance Costs," *Comm. ACM*, vol. 36, no. 11, pp. 81-94, 1993.

[55] R.D. Banker, G.B. Davis, and S.A. Slaughter, "Software Development Practices, Software Complexity and Software Maintenance Performance: A Field Study," *Management Science*, vol. 44, pp. 433-450, 1998.

[56] J.E. Gaffney, "Estimating the Number of Faults in Code," *IEEE Trans. Software Eng.*, vol. 10, no. 4, pp. 459-465, 1984.

[57] V.Y. Shen, T. Yu, and S.M. Thebut, "Identifying Error-Prone Software – An Empirical Study," *IEEE Trans. Software Eng.*, vol. 11, pp. 317-324, 1985.

[58] R.D. Banker and C.F. Kemerer, "Scale Economies in New Software Development," *IEEE Trans. Software Eng.*, pp. 1199-1205, 1989.

[59] R. Subramanyam, M.S. Krishnan, "Empirical Analysis of CK Metrics for Object-Oriented Design Complexity: Implications for Software Defects," *IEEE Trans. Software Eng.*, vol. 29, no. 4, pp. 297-310, 2003.

[60] F. Brooks, *The Mythical Man Month*, Anniversary Edition, 1995.

[61] E.T.G. Wang, P. Ju, J.J. Jiang, G. Klein, "The Effects of Change Control and Management Review on Software Flexibility and Project Performance," Information and Management, vol. 45, no. 7, pp. 438, 2008.

[62] D. Damian, J. Chisan, "An Empirical Study of the Complex Relationships between Requirements Engineering Processes and Other Processes that Lead to Payoffs in Productivity, Quality, and Risk Management," *IEEE Trans. Software Eng.*, vol. 32, no. 7, pp. 433-453, 2006.

[63] K. El Emam, *The ROI from Software Quality*, Auerbach Publications, 2005.

[64] D.E. Harter, and S. Slaughter, "Quality Improvement and Infrastructure Activity Costs in Software Development: A Longitudinal Analysis," *Management Science*, vol. 49, no. 6. pp. 784-800, 2003.

[65] T.F. Liao, *Interpreting Probability Models: Logit, Probit, and Other Generalized Linear Models*, Sage Publications, 1994.

[66] W.H. Greene, *Econometric Analysis*, 3rd Edition, MacMillan Publishing Company, New York, 1997.

[67] C. Ai, and E.C. Norton, "Interaction Terms in Logit and Probit Models," *Economics Letters*, vol. 80 pp. 123-129, 2003.

[68] E.C. Norton, H. Wang, C. Ai, "Computing Interaction Effects and Standard Errors in Logit and Probit Models," *The Stata Journal*, vol. 4, no. 2, pp. 154-167.

First A. Author Biographies should be limited to one paragraph consisting of the following: sequentially ordered list of degrees, including years achieved; sequentially ordered places of employ concluding with current employment; association with any official journals or conferences; major professional and/or academic achievements, i.e., best paper awards, research grants, etc.; any publication information (number of papers and titles of books published); current research interests; association with any professional associations.

Second B. Author Jr. biography appears here. Degrees achieved followed by current employment are listed, plus any major academic achievements.

Third C. Author is a member of the IEEE and the IEEE Computer Society.

Table 1: Summary of Research Questions

1	What effect do investments in CMM software process improvements have on the severity of software defects? Does advancing to a higher CMM level have a differential impact on the likelihood of critical versus minor defects? What implications do these distinctions have for software development?
2	How do higher levels of software process improvement differentially affect defect severity by life cycle stage? Are the effects on development and production defects different, and, if so, how should this knowledge affect investment rates and plans for software improvement?
3	What types of systems exhibit the highest benefits in terms of a reduced likelihood of severe defects from improvements in software processes? Do the size, complexity and requirements ambiguity of the system affect the improvement in defect severity, and, if so, how should this influence management decisions?

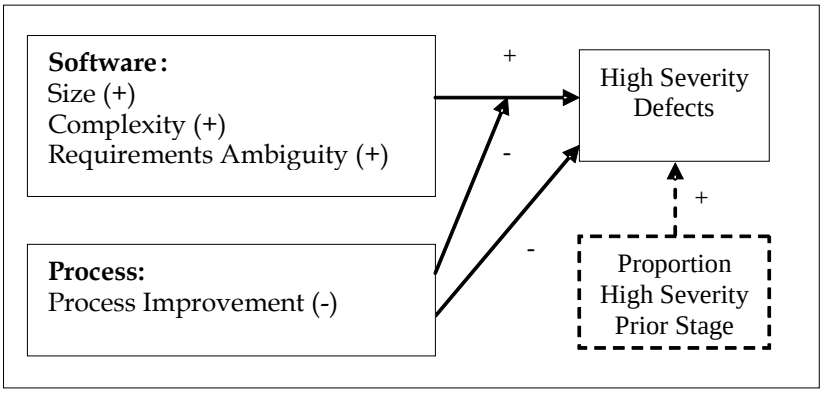


Figure 1. Development and Production Defect Severity Models

TABLE 2
Software Measures

Factor	Description (Measure)
High Severity Defect	Major functional failure or abnormal termination (1)
Low Severity Defect	Minor or cosmetic error (0)
Software Process Improvement	The SEI Capability Maturity Model (CMM) level at which the software was developed (ranging from 1 to 5)
Software Size	Thousands of Lines of Code (KLOC) of the software
Software Complexity	Domain, data, and decision complexity of the software (rated on scales of 1 to 5) [24]
Requirements Ambiguity	Level of ambiguity in the requirements specification for the software (rated on a scale of 1 to 5) [24]

TABLE 3
Domain Complexity Measures [24]

Domain Complexity	Description
1	Simple algorithms and simple calculations
2	Majority of simple algorithms and simple calculations
3	Algorithms and calculations of average complexity
4	Some difficult algorithms or complex calculations
5	Many difficult algorithms and complex calculations

TABLE 4
Data Complexity Measures [24]

Data Complexity	Description
1	Simple data with few variables and low complexity
2	Several data elements, but simple data relationships
3	Multiple files, data interactions, and file updates
4	Complex file structures, data interactions, and updates
5	Very complex data elements, interactions, and updates

TABLE 5
Decision Complexity Measures [24]

Decision Complexity	Description
1	Non-procedural (e.g., spreadsheet, query language)
2	Well-structured, plus standard reusable modules
3	Well-structured, with small modules and simple paths
4	Fair structure, but some complex modules and paths
5	Poor structure, with many complex modules and paths

TABLE 6
Requirements Ambiguity Measures [24]

Requirements Ambiguity	Description
1	Program developers are also program users
2	Working model or prototype, plus clear requirements
3	Fairly clear user requirements
4	Ambiguous or incomplete user requirements
5	Ambiguous, incomplete, and rapidly changing user requirements

TABLE 7
Data Collection Summary

Data	Collected by	Audited by
Software Process Improvement (CMM Maturity Level)	• External divisions • Client's auditors	• External senior executives • Client's auditors
Software Size	• Configuration Management	• Quality Assurance • Client technical reviewers
Software Complexity	• Software engineering manager	• Chief Engineer • Client technical reviewers
Requirements Ambiguity	• Software engineering manager	• Chief Engineer • Client technical reviewers
Development Defect Severity	• Configuration Management	• Quality Assurance • Client technical reviewers
Production Defect Severity	• Configuration Management	• Quality Assurance • Client technical reviewers

TABLE 8
Development Defect Severity Summary Statistics by Defect

Variable	Mean	Standard Deviation	Minimum	Median	Maximum
1. Process Improvement			1.00	2.00	5.00
2. Software Size (KLOC)	175.42	148.27	0.17	129.41	605.58
3. Complexity			2.23	3.15	4.00
4. Requirements Ambiguity			1.00	3.00	5.00

TABLE 9
Production Defect Severity Summary Statistics by Defect

Variable	Mean	Standard Deviation	Minimum	Median	Maximum
1. Process Improvement			1.00	2.16	5.00
2. Software Size (KLOC)	67.22	68.64	0.17	45.13	312.62
3. Complexity			2.23	3.15	4.00
4. Requirements Ambiguity			1.00	3.10	5.00
5. Percent of High Severity in Development	0.58	0.23	0.00	0.59	1.00

TABLE 10
Development Defect Severity Correlation Matrix by Defect
(n=6,190; Pearson correlation coefficients with *p-values* in parentheses)

	1. High Severity	2. Process Improvement	3. Software Size	4. Design Complexity	5. Requirements Ambiguity
1. High Severity	1.0000				
2. Process Improvement	-0.0086 (0.4984)	1.0000			
3. Software Size	-0.0070 (0.5824)	0.2589 (0.0000)	1.0000		
4. Complexity	0.0155 (0.2225)	0.5839 (0.0000)	0.2329 (0.0000)	1.000	
5. Requirements Ambiguity	0.0493 (0.0001)	0.0378 (0.0029)	-0.1714 (0.0000)	0.2209 (0.0000)	1.0000

TABLE 11
Production Defect Severity Correlation Matrix by Defect
(n=1,355; Pearson correlation coefficients with p values in parentheses)

	1. High Severity	2. Process Improvement	3. Software Size	4. Design Complexity	5. Requirements Ambiguity	6. Percent Devel. High Severity
1. High Severity	1.0000					
2. Process Improvement	-0.1533 (0.0000)	1.0000				
3. Software Size	0.0445 (0.1042)	-0.4432 (0.0000)	1.0000			
4. Complexity	-0.0601 (0.0281)	0.4939 (0.0000)	-0.1022 (0.0002)	1.0000		
5. Requirements Ambiguity	0.0880 (0.0013)	0.1087 (0.0001)	-0.0160 (0.5591)	0.3413 (0.0000)	1.0000	
6. Percent Development High Severity	0.0564 (0.0394)	0.0707 (0.0098)	-0.0308 (0.2610)	0.4075 (0.0000)	0.0946 (0.0005)	1.0000

TABLE 12
Development Severity Probit Parameter Estimates (n=6190) (one-tailed p-values)

Variable	Parameter	Controls	Controls + CMM	Controls + CMM + Interactions
Intercept	β_{01}	**0.202**	**0.202**	**0.275**
	p	0.000	0.000	0.000
Size	β_{11}	**0.000**	**0.000**	**0.001**
	p	0.497	0.413	0.000
Complexity	β_{21}	**0.018**	**0.063**	**-0.033**
	p	0.361	0.154	0.304
Ambiguity	β_{31}	**0.104**	**0.101**	**0.115**
	p	0.000	0.001	0.000
Process-Improvement (CMM)	β_{41}		**-0.046**	**-0.069**
	p		0.097	0.032
CMM * Size	β_{51}			**-0.001**
	p			0.000
CMM * Complexity	β_{61}			**-0.370**
	p			0.000
CMM * Ambiguity	β_{71}			**0.142**
	p			0.002
Wald Statistic	χ^2	**15.16**	**16.85**	**84.59**
	P	0.002	0.002	0.000

TABLE 13
Production Severity Probit Parameter Estimates (n=1336) (one-tailed p-values)

Variable	Parameter	Controls	Controls + CMM	Controls + CMM + Interactions
Intercept	β_{02}	**0.309**	**0.313**	**0.348**
	p	0.000	0.000	0.000
Percent Devel. High Severity	β_{12}	**0.581**	**0.437**	**0.517**
	p	0.001	0.005	0.002
Size	β_{22}	**0.001**	**-0.001**	**-0.001**
	p	0.111	0.185	0.041
Complexity	β_{32}	**-0.437**	**-0.179**	**-0.122**
	p	0.000	0.059	0.150
Ambiguity	β_{42}	**0.236**	**0.219**	**0.218**
	p	0.000	0.000	0.000
Process-Improvement (CMM)	β_{52}		**-0.299**	**-0.370**
	p		0.000	0.000
CMM * Size	β_{62}			**-0.002**
	p			0.047
CMM * Complexity	β_{72}			**-0.492**
	p			0.001
CMM * Ambiguity	β_{82}			**0.005**
	p			0.481
Wald Statistic	χ^2	**36.44**	**55.07**	**71.00**
	P	0.000	0.000	0.000

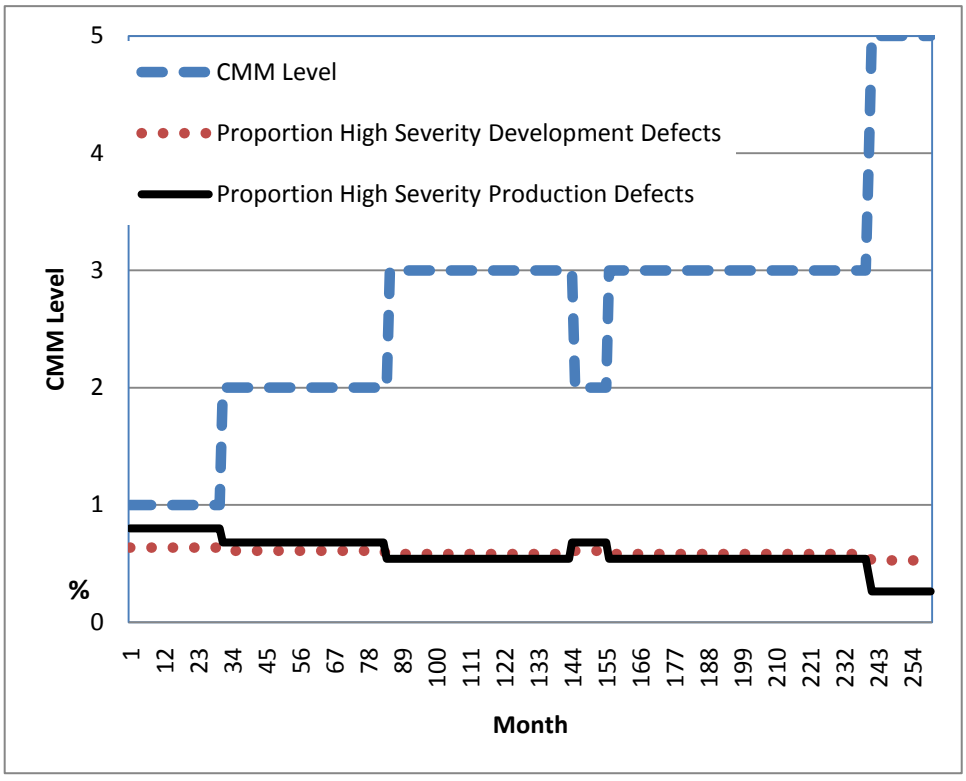
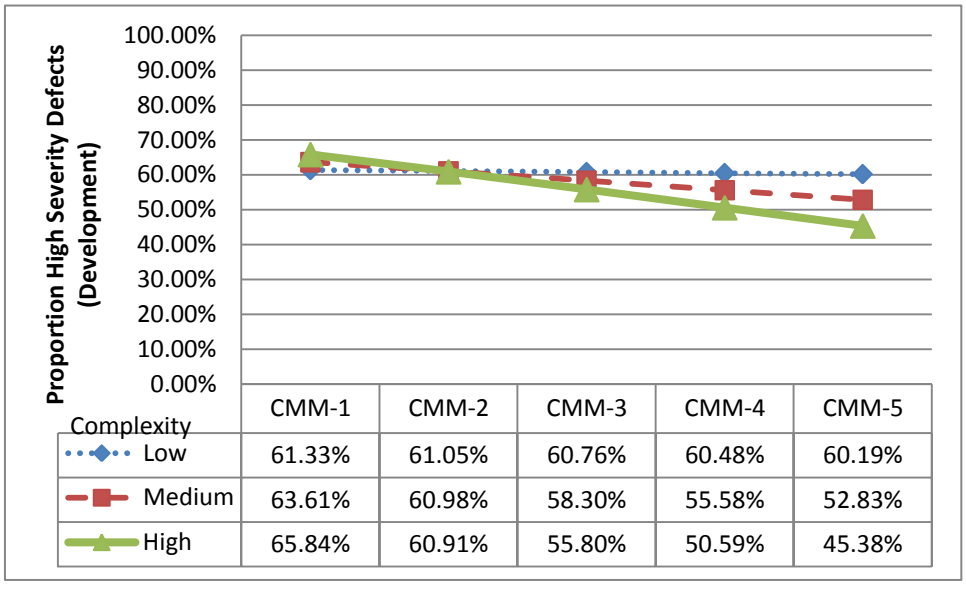
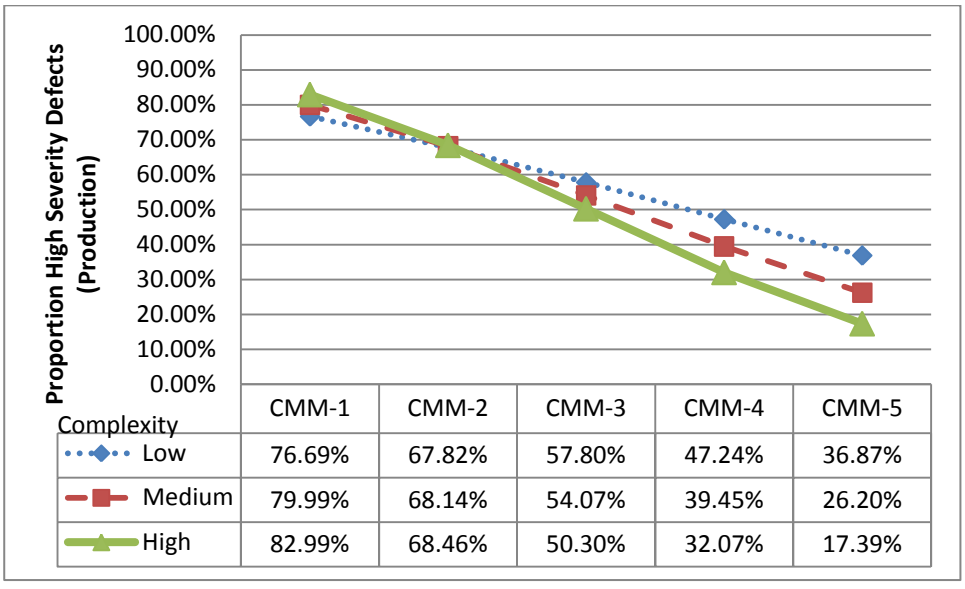


Figure 2. Process Improvements and Defect Severity Over Time



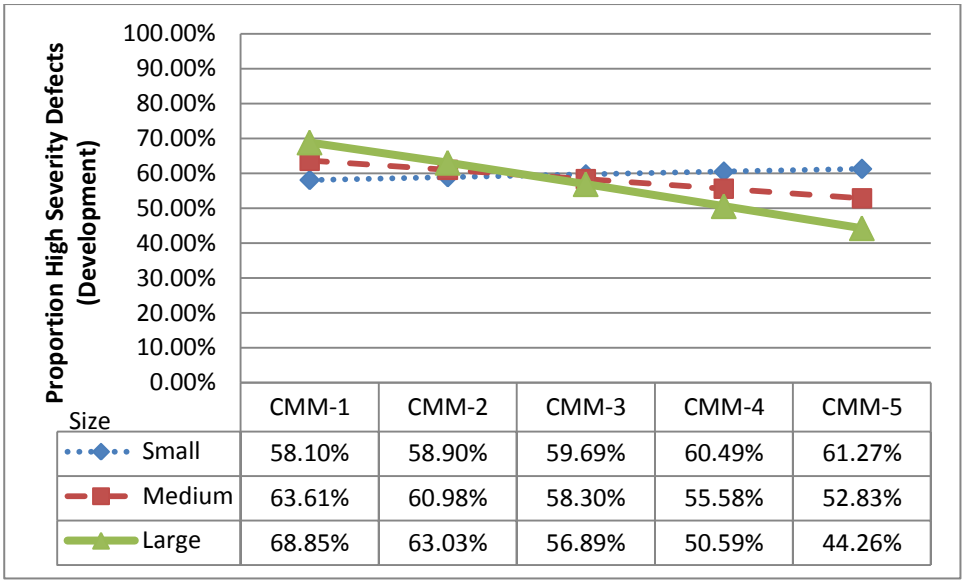
Complexity	CMM-1	CMM-2	CMM-3	CMM-4	CMM-5
Low	61.33%	61.05%	60.76%	60.48%	60.19%
Medium	63.61%	60.98%	58.30%	55.58%	52.83%
High	65.84%	60.91%	55.80%	50.59%	45.38%

Figure 3. Interaction Effect of CMM and Complexity on Development Defects



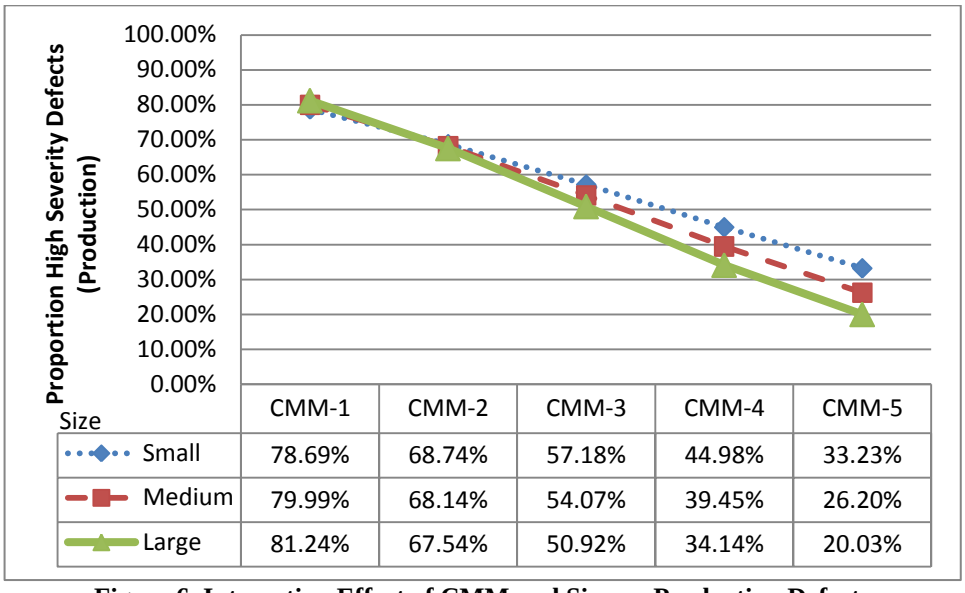
Complexity	CMM-1	CMM-2	CMM-3	CMM-4	CMM-5
Low	76.69%	67.82%	57.80%	47.24%	36.87%
Medium	79.99%	68.14%	54.07%	39.45%	26.20%
High	82.99%	68.46%	50.30%	32.07%	17.39%

Figure 4. Interaction Effect of CMM and Complexity on Production Defects



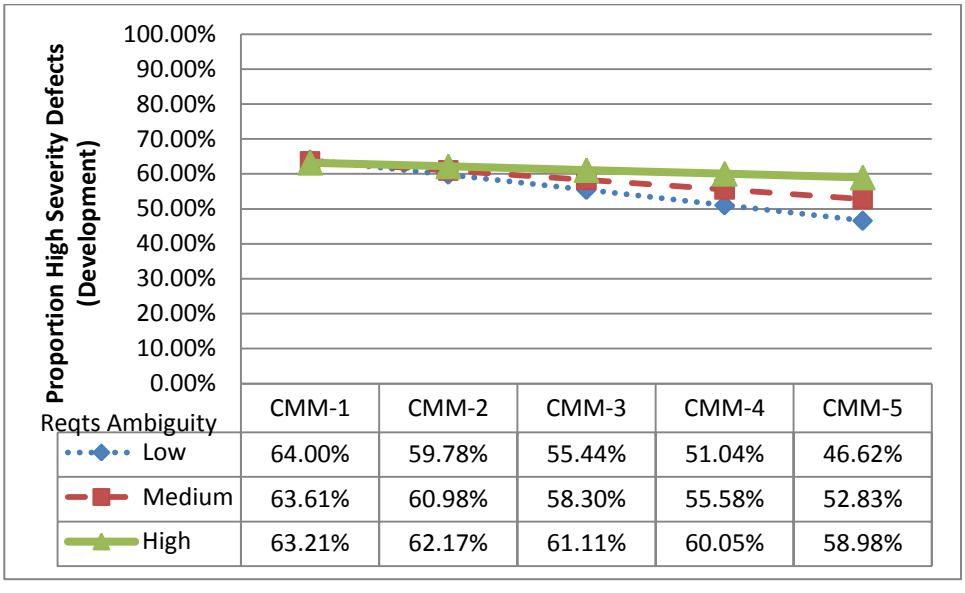
Size	CMM-1	CMM-2	CMM-3	CMM-4	CMM-5
Small	58.10%	58.90%	59.69%	60.49%	61.27%
Medium	63.61%	60.98%	58.30%	55.58%	52.83%
Large	68.85%	63.03%	56.89%	50.59%	44.26%

Figure 5. Interaction Effect of CMM and Size on Development Defects



Size	CMM-1	CMM-2	CMM-3	CMM-4	CMM-5
Small	78.69%	68.74%	57.18%	44.98%	33.23%
Medium	79.99%	68.14%	54.07%	39.45%	26.20%
Large	81.24%	67.54%	50.92%	34.14%	20.03%

Figure 6. Interaction Effect of CMM and Size on Production Defects



Reqts Ambiguity	CMM-1	CMM-2	CMM-3	CMM-4	CMM-5
Low	64.00%	59.78%	55.44%	51.04%	46.62%
Medium	63.61%	60.98%	58.30%	55.58%	52.83%
High	63.21%	62.17%	61.11%	60.05%	58.98%

Figure 7. Interaction Effect of CMM and Ambiguity on Development Defects

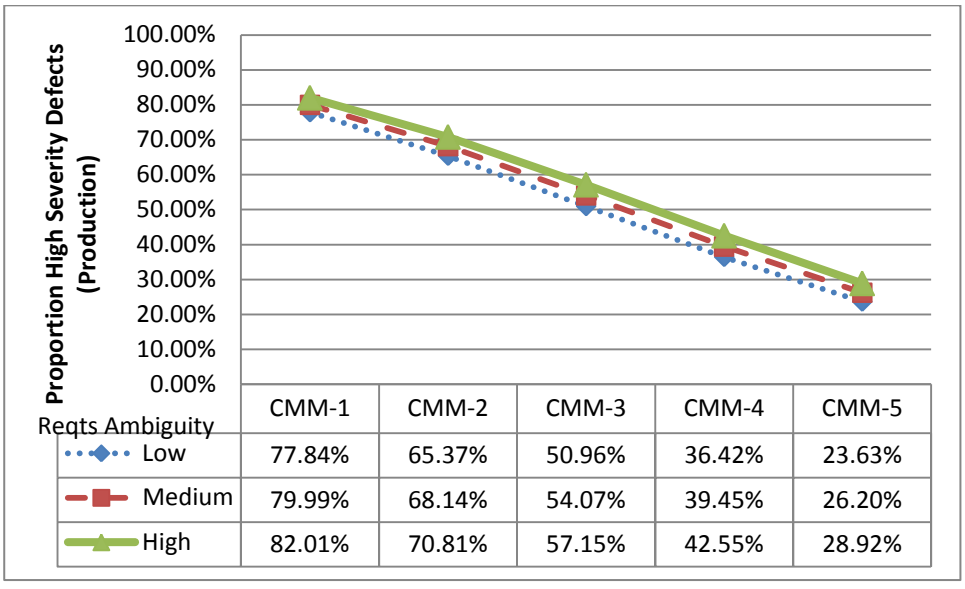


Figure 8. Interaction Effect of CMM and Ambiguity on Production Defects

TABLE 14
Empirical Support for Hypotheses by Model (α=.05)

Variable	Development Model	Production Model
Process (main effect)	**H1**: Supported	**H2**: Supported
Complexity (interaction effect)	**H3**: Supported	**H3**: Supported
Size (interaction effect)	**H4**: Supported	**H4**: Supported
Requirements Ambiguity (interaction effect)	**H5**: Supported	**H5**: Not Supported