

USING ERROR MODELING TO IMPROVE AND CONTROL SOFTWARE
QUALITY: AN EMPIRICAL INVESTIGATION

By

Gursimran Singh Walia

A Dissertation
Submitted to the Faculty of
Mississippi State University
in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy
in Computer Science
in the Department of Computer Science and Engineering

Mississippi State, Mississippi

May 2009

Copyright by
Gursimran Singh Walia
2009

USING ERROR MODELLING TO IMPROVE AND CONTROL SOFTWARE
QUALITY: AN EMPIRICAL INVESTIGATION

By

Gursimran Singh Walia

Approved:

Jeffrey C. Carver
Adjunct Assistant Professor of Computer
Science and Engineering
(Major Professor)

Edward B. Allen
Associate Professor of Computer Science
and Engineering, and
Graduate Coordinator
(Committee Member)

Thomas Philip
Professor of Computer Science and
and Engineering
(Committee Member)

Gary L. Bradshaw
Professor of Psychology
(Committee Member)

Sarah A. Rajala
Dean of the James Worth Bagley
College of Engineering

Name: Gursimran Singh Walia

Date of Degree: May 2, 2009.

Institution: Mississippi State University

Major Field: Computer Science

Major Professor: Jeffrey C. Carver

Title of Study: USING ERROR MODELLING TO IMPROVE AND CONTROL
SOFTWARE QUALITY: AN EMPIRICAL INVESTIGATION

Pages in Study: 148

Candidate for Degree of Doctor of Philosophy

Software quality, or the lack thereof, is a well-known problem faced by software engineers. To address the problem of poor software quality, many approaches have been developed and evaluated through controlled experiments and case studies in both laboratory and real settings. Considerable effort has been devoted to identify methods to find and repair early life cycle faults i.e., actual mistakes recorded in a requirement or design document. One empirically proven fault-reduction approach is to use fault classification taxonomies to help developers identify different types of important faults. However, even when faithfully applying various empirically-validated techniques that allow developers to focus on faults, they do not help developers to find all types of problems. Furthermore, these techniques can only help detect the presence of faults and not their absence or provide insight into how many faults still remain. To augment the existing methods, and further improve software quality, my research developed and validated new approaches to fill in some of the gaps.

This dissertation involves developing and validating effective methods and tools for improving and measuring the quality of software artifacts. A major focus of this dissertation is to understand the thought process of developers so that software quality can be improved. Another

focus of my research is to support software defect estimation post-inspection to manage software quality. This dissertation exploits the knowledge about software development errors (i.e., source of the faults) and using the error information to develop techniques that will help developers find and eliminate errors early in the software lifecycle process.

This dissertation research is multidisciplinary as it uses approaches that have been applied successfully in other domains and adapts them for the task of improving and managing software quality. This dissertation research added information from cognitive psychology research about human errors to extend taxonomies of software development errors, and also use the Capture-Recapture method (used by biologists and wildlife researchers) to support defect size estimates of software artifacts to manage the software inspection process.

DEDICATION

I would like to dedicate this research to my friends and family.

ACKNOWLEDGMENTS

I want to thank Dr. Jeffrey C. Carver for guiding me through this research, for always encouraging me to do better, and in writing this dissertation.

I would like to thank Dr. Thomas Philip, and Dr. Jeffrey C. Carver for allowing me to conduct the experiments in their courses and providing me the opportunities to present and discuss my research in various courses that was helpful to further my research ideas. I also thank Dr. Gary Bradshaw for providing valuable sources of research into human cognition and psychology, and his critique in presenting my research. I would like to thank all my committee members for providing valuable suggestions to this ongoing dissertation work.

I also thank all the members of Empirical Software Engineering group at Mississippi State University for providing valuable feedback and suggestions, and providing me ample opportunities to present my research. Also, I want to thank all the students at Mississippi State University that participated in the experiments.

I also thank Nachiappan Nagappan at Microsoft Corporation for providing me with the inspection data from Microsoft that helped me in this dissertation work.

TABLE OF CONTENTS

	Page
DEDICATION	ii
ACKNOWLEDGMENTS.....	iii
LIST OF TABLES	vii
LIST OF FIGURES.....	x
CHAPTER	
I. INTRODUCTION.....	1
1.1 Improving Software Quality.....	1
1.2 Managing Software Quality	5
1.3 Research Hypotheses.....	7
1.4 Framework of Research Activities.....	7
1.5 Organization.....	8
II. LITERATURE REVIEW.....	9
2.1 Research on Sources of Faults	10
2.2 A Cognitive Psychology Perspective on Errors	11
2.3 Related Work on the Use of Error Information to Improve Software Quality.....	14
2.4 Use of Capture-Recapture in Software Inspections	16
2.4.1 Capture-Recapture Models and Estimators.....	16
2.4.2 Empirical Studies on Capture-Recapture in Software Inspections.....	19
III. RESEARCH APPROACH.....	23
3.1 Research Method for Developing Requirement Error Taxonomy	23
3.2 Description of the Systematic Review Process	24
3.2.1 Research Questions	25
3.2.2 Data Extraction and Synthesis.....	30
3.3 Reporting the Review: Answering Research Questions	30
3.4 Developing the Requirement Error Taxonomy.....	42

3.4.1	Developing the Requirement Error Classes	43
3.4.2	Developing the Requirement Error Types	51
3.5	Empirically Validating the Requirement Error Taxonomy.....	55
3.6	Empirically Validating the Use of Capture-Recapture in Software Inspections.....	58
IV.	EMPIRICAL VALIDATION OF THE REQUIREMENT ERROR TAXONOMY	62
4.1	Study Hypotheses and Variables.....	62
4.2	Study 2: A Control Group Replicated Study.....	64
4.2.1	Study Design	64
4.2.2	Data Analysis and Results.....	66
4.2.2.1	Hypothesis 1: Improve Effectiveness and Efficiency ..	66
4.2.2.2	Hypothesis 2: Usefulness of Requirement Error Taxonomy.....	67
4.2.2.3	Hypothesis 3: Insights into the Major Source (s) of Requirement Faults	68
4.2.2.4	Hypothesis 4: Usefulness of Cognitive Psychology Research	70
4.2.2.5	Hypothesis 5: Effect of Independent Variables.....	70
4.2.3	Validity Threats.....	71
4.2.4	Results Summary.....	73
4.3	Study 1: Feasibility Study and Study 3: An Observational Study	73
4.3.1	Study Designs.....	73
4.3.2	Data Analysis and Results.....	76
4.3.2.1	Hypothesis 1: Improve Effectiveness and Efficiency ..	76
4.3.2.2	Hypothesis 2: Usefulness of Requirement Error Taxonomy.....	78
4.3.2.3	Hypothesis 3: Insights into the Major Source (s) of Requirement Faults	79
4.3.2.4	Hypothesis 4: Usefulness of Cognitive Psychology Research	80
4.3.2.5	Hypothesis 5: Effect of Independent Variables.....	81
4.3.3	Validity Threats.....	85
4.3.4	Summary of Results	85
4.4	Discussion of Results from Three Studies	85
4.4.1	Hypothesis 1	86
4.4.2	Hypothesis 2	87
4.4.3	Hypothesis 3	89
4.4.4	Hypothesis 4.....	90
4.4.5	Hypothesis 5.....	91
4.4.6	Validity Threats in All Three Studies	91
4.4.7	Results Summary.....	92

V.	EMPIRICAL VALIDATION OF CAPTURE RECAPTURE IN SOFTWARE INSPECTIONS	93
5.1	Study 1: Investigating the Effect of the Number of Inspectors.....	93
5.1.1	Data Set	94
5.1.2	Experiment Procedure	94
5.1.3	Evaluation Criterion	96
5.1.4	Analysis and Results	96
5.1.5	Validity Threats.....	102
5.2	Study 2: Investigating the Effect of the Number of Faults	103
5.2.1	Data Set	104
5.2.2	Experiment Procedure	105
5.2.3	Evaluation Criterion	105
5.2.4	Analysis and Results	106
5.2.5	Validity Threats.....	116
5.3	Study 3: Evaluating Capture Recapture Models and Estimators for Estimating the Abundance of Naturally Occurring Faults	117
5.3.1	Data Set	118
5.3.2	Experiment Procedure	119
5.3.3	Analysis and Results	119
	5.3.3.1 Comparison of Estimates After First and Second Inspection	120
	5.3.3.2 Selection of the Best Capture Recapture Model	122
	5.3.3.3 Using Capture Recapture to Manage the Inspection Process.....	124
5.3.4	Validity Threats.....	128
5.4	Dicussion of Results.....	128
5.4.1	Summary of Major Findings from Study 1	129
5.4.2	Summary of Major Findings from Study 2	129
5.4.3	Summary of Major Findings from Study 3	130
VI.	IMPORTANCE OF RESULTS	132
VII.	CONCLUSION	135
7.1	Contribution to Research and Practice Communities	135
7.2	Summary	136
7.2	Publications	137
7.2	Future Research Directions	138
	REFERENCES.....	140

LIST OF TABLES

TABLE	Page
2.1 CAPTURE-RECAPTURE MODELS [17]	17
2.2 CAPTURE-RECAPTURE ESTIMATORS	18
3.1 RESEARCH QUESTIONS AND MOTIVATIONS.....	26
3.2 INCLUSION AND EXCLUSION CRITERIA	28
3.3 PAPER DISTRIBUTION.....	29
3.4 LIMITATIONS OF METHODS IN QUESTION 1.1	34
3.5 SOURCES USED TO IDENTIFY REQUIREMENT ERRORS IN THE LITERATURE.....	35
3.6 REQUIREMENT ERRORS DRAWN FROM HUMAN ERRORS	41
3.7 COMMUNICATION ERRORS.....	44
3.8 PARTICIPATION ERRORS.....	45
3.9 DOMAIN KNOWLEDGE ERRORS.....	45
3.10 SPECIFIC APPLICATION ERRORS.....	46
3.11 PROCESS EXECUTION ERRORS.....	46
3.12 OTHER HUMAN COGNITION ERRORS	47
3.13 INADEQUATE METHOD OF ACHIEVING GOALS AND OBJECTIVES ERRORS.....	47
3.14 MANAGEMENT ERRORS.....	48
3.15 REQUIREMENT ELICITATION ERRORS	48

3.16	REQUIREMENT ANALYSIS ERRORS	49
3.17	REQUIREMENT TRACEABILITY ERRORS	49
3.18	REQUIREMENT ORGANIZATION ERRORS.....	50
3.19	NO USE OF STANDARD FOR DOCUMENTING ERRORS.....	50
3.20	SPECIFICATION ERRORS	51
3.21	REQUIREMENT ERROR TYPES	52
4.1	STUDY HYPOTHESES.....	63
4.2	INDEPENDENT AND DEPENDENT VARIABLES.....	63
4.3	STEPS PERFORMED BY EXPERIMENT GROUP AND CONTROL GROUP.....	65
4.4	INSIGHTS PROVIDED BY THE REQUIREMENT ERROR TAXONOMY	68
4.5	EFFECT OF INDEPENDENT VARIABLES ON SUBJECT'S EFFECTIVENESS.....	71
4.6	SUMMARY OF FINDINGS FROM STUDY 2	72
4.7	STUDY 1 AND STUDY 3: TEAMS, SUBJECTS, AND SYSTEM DESCRIPTION	74
4.8	DIFFERENCES IN STUDY 1 AND STUDY 3 FROM STUDY 2.....	74
4.9	CONTRIBUTION OF ERROR TYPES TO OVERALL ERRORS AND FAULTS	79
4.10	ADDITIONAL INSIGHTS PROVIDED BY THE REQUIREMENT ERROR TAXONOMY	80
4.11	SUMMARY OF FINDINGS FROM STUDY 1 AND STUDY 3	86

4.12	INSIGHTS PROVIDED BY THE REQUIREMENT ERROR TAXONOMY	88
4.13	SUMMARY OF FINDINGS FROM ALL THE STUDIES.....	92
5.1	CUT OFF POINTS FOR REGIONS 1, 2, AND 3	101
5.2	MINIMUM NUMBER OF INSPECTORS FOR DIFFERENT LEVELS OF PERFORMANCE.....	102
5.3	REQUIREMENT ARTIFACTS AND INSPECTORS USED IN STUDY 2	104
5.4	PERCENTAGE OF FAULTS REQUIRED FOR DIFFERENT LEVELS OF ESTIMATION ACCURACY.....	109
5.5	CORRECTNESS OF RE-INSPECTION DECISIONS FOR ARTIFACTS A AND B	114
5.6	ARTIFACTS, AND INSPECTORS USED IN STUDY 3	118

LIST OF FIGURES

FIGURE	Page
2.1 CR Data Input Matrix	19
3.1 Types of Literature Searched to Identify the Requirement Errors.....	25
3.2 Requirement Error Taxonomy	54
3.3 This Sequence and Design of Three Controlled Experiments Studies to Evaluate the Requirement Error Taxonomy.....	56
3.4 Framework of the Studies Conducted to Evaluate the Capture-Recapture Models and Estimators	59
4.1 Methodology for Study 2	64
4.2 Average Number of Faults Found by Experiment and Control Group at 1 st , 2 nd , and Total Inspection	66
4.3 Contribution of Cognitive Psychology to Overall Error and Faults found by Eight Subjects in the Experiment Group.....	70
4.4 Study Operation for Study 1 in Solid Lines and the Additional Observational Aspect in Study 3 in Dotted Lines	75
4.5 Increase in the Number of Faults Found	77
4.6 Contribution of Cognitive Psychology Research to Overall Error and Faults Found by Each Subject in Teams 1-A, 1-B, 3-A, and 3-B.....	81
5.1 Total Fault(s) Found by Each Inspector.....	95
5.2 Median Estimate as a Function of Number of Inspectors for All CR Estimators.....	97
5.3 Relative Error Variability as a Function of the Number of Inspectors for All CR Estimators	98

5.4	Variations of M_0 -CMLE Estimates as a Function of the Number of Inspectors	99
5.5	Variations of M_h -JK Estimates as a Function of the Number of Inspectors	100
5.6	Median Relative Error Values for Different Fault Counts for Artifact A.....	106
5.7	Median Relative Error Values for Different Fault Counts for Artifact B.....	107
5.8	Variability in the Estimates for Different Fault Counts for Artifact B	108
5.9	Relative Estimate of Remaining Faults of All Counts for Artifact A.....	110
5.10	Relative Estimate of Remaining Faults of All Counts for Artifact B	111
5.11	Comparisons of Estimate and Actual Relative Error for M_h -SC on Artifact A.....	113
5.12	Comparisons of Estimate and Actual Relative Error for M_h -SC on Artifact B	114
5.13	Remaining Faults at Different Percentage of Total Faults for Both Artifacts ..	115
5.14	Relative Errors in Estimates after First Inspection.....	120
5.15	Relative Errors in Estimates after Second Inspection	121
5.16	Estimated Remaining Faults after the First Inspection	126
5.17	Estimated Remaining Faults after the Second Inspection.....	127

CHAPTER I

INTRODUCTION

This dissertation addresses two well-known problem faced by software engineers: *Software Quality Improvement* and *Software Quality Measurement*. My dissertation efforts involved developing and empirically validating effective methods and tools to improve and measure the quality of software artifacts. This dissertation work is multidisciplinary in that it exploits knowledge from other research domains (Cognitive Psychology and Wildlife Science) to help address the problem of improving and managing software quality. The purpose of this chapter is to outline the problem statement, clarify terminology that will be used in this dissertation, present the research motivation, and outlines the research hypotheses and activities.

1.1 Improving Software Quality

Software quality, or the lack thereof, is a well-known problem faced by software engineers. To address this problem, many approaches have been developed and evaluated through controlled and case studies (e.g. [21, 34, 50]). Considerable effort has been devoted to identifying methods to find and repair problems early in the software lifecycle when these repairs are easiest and cheapest. The goal of these methods is to detect and remove early-lifecycle faults i.e., mistakes recorded in a requirements or design artifact.

To accomplish this goal, *fault classification taxonomies* help software developers identify different types of important faults. The value of using such an approach has been established by multiple empirical studies (e.g. [11, 12, 21, 75]). However, even when faithfully applying various empirically-validated fault-based techniques, software quality is still not at the desired level. It is estimated that 40-50% of effort is spent on avoidable rework, that is fixing problems that should have been fixed earlier in the lifecycle, or should have been prevented. Much of this rework is the result of the fact that early lifecycle fault detection techniques are based on incomplete fault taxonomies and do not lead developers to find all types of problems. Therefore, there is still room for significant improvement in early lifecycle defect detection and removal to eliminate some, or all, of the unnecessary rework. This dissertation presents an approach that augments existing fault taxonomies and further improves software quality.

Before discussing software quality any further, it is important to clarify a few important terms: *error*, *fault*, and *failure*. Unfortunately, the software engineering literature contains competing and often contradictory definitions of these terms. In fact, IEEE standard 610.12-1990 provides four definitions of the terms *error*, ranging from “incorrect program condition” (referred to as a *program error*) to “mistake in the human thought process” (referred to as a *human error*) [1]. To allay confusion, we provide a definition for each term that will be used consistently throughout this dissertation. These definitions were originally given by Lanubile, et al. [49], and are consistent with software engineering textbooks [31, 64, 72] and IEEE Standard 610.12-1990 [1]:

- *Error* – defect in the human thought process made while trying to understand given information, solve problems, or to use methods and tools. In the context

of software requirements specifications, an error is a basic misconception of the actual needs of a user or customer.

- *Fault* – concrete manifestation of an error within the software. One error may cause several faults, and various errors may cause identical faults.
- *Failure* – departure of the operational software system behavior from user expected requirements. A particular failure may be caused by several faults and some faults may never cause a failure.

The term *defect* is a generic term used to describe any of these three types of problems.

The definition of an error used in this dissertation more closely relates to the *human error* definition rather than the *program error* definition in IEEE Standard 610.12-1990.

The distinction among errors and faults is important in ensuring software quality. As indicated in the previous paragraphs, most quality improvement efforts have been driven by fault taxonomies and techniques derived from those taxonomies. The main drawback of this approach is the absence of a direct mapping between errors (the source of the problem) and faults (the manifestation of the problem). For example, a common requirement fault is the omission of one or more requirements that describe important functionality. This type of omission has at least two possible sources: 1) the requirement author might have lacked the proper domain knowledge to realize that the requirement(s) were needed, or 2) important stakeholders were left out of the requirement elicitation process and therefore their needs were not recorded in the requirement document. In order to find and fix the real problem, it is important for the development team to determine not only that the omission occurred, but more importantly why it occurred.

This dissertation argues that an error-based approach is an important next step for improving software quality. If guidance can be provided to developers to help them

identify and eliminate errors, then consequently, the number of faults will decrease and software quality will increase. Furthermore, by identifying errors that have occurred, developers can then find additional related faults that may have otherwise been overlooked (similar to the way a doctor will find and treat all symptoms once he determines the underlying disease). A taxonomy created by organizing the errors can provide information to help developers detect errors in the same way that fault taxonomies are used to help identify faults.

Because software engineering is a human-based activity, it is reasonable to investigate phenomena associated with the human mental process and its fallibilities in relation to the development of software artifacts. This dissertation exploits advances that have been made by cognitive psychologists in understanding *human errors*. Psychological research draws upon models of human reasoning, planning, and problem solving, and how these ordinary psychological processes can go awry. Exploiting a connection with cognitive research is particularly useful for understanding the cause of software faults because case studies have shown that errors related to everyday human cognitive processes, not necessarily related to software engineering, can creep into the software development process (e.g., [51]). Therefore, the major goal of this dissertation is to integrate research findings from cognitive psychology with findings from software quality improvement, facilitated by an in-depth understanding of how the human cognitive process can fail as a developer creates various software artifacts.

To produce a list of errors, and make that information useful for quality improvement, this dissertation has two major thrusts (each of which will be more fully discussed in Chapter III and Chapter IV respectively):

- Combine generic human error research from cognitive psychology with specific knowledge of faults and errors from software engineering to develop a *Requirement Error Taxonomy*, and
- Empirically validate the *Requirement Error Taxonomy* for feasibility, completeness, and usefulness to software developers in improving quality of software artifacts.

1.2 Managing Software Quality

The software development process involves the translation of information from one form to another (i.e., from customers needs to requirements to architecture to design to code). Project managers and software developers manage the software development process by monitoring the quality of artifacts developed at each lifecycle stage.

Using an error-based approach can improve the effectiveness of the inspection process by helping developers locate defects early and prevent their propagation to subsequent phases. However, inspections can only certify the presence of defects, not their absence. Therefore, inspections alone do not provide all the information needed to reason about the overall quality of a software artifact. Furthermore, evidence suggests that the effectiveness of an inspection varies widely because defects can be overlooked during an inspection and remain undetected [17].

So, for any live development project that contains an unknown number of defects, determining the artifact's quality is a difficult and important problem. To gain a better

understanding of the quality of their software, project managers need objective information to help them decide when they can safely stop the inspection process. An informed, objective estimate of remaining defects should guide this decision rather than subjective opinion or historical trends. This dissertation research investigates the use of the *capture-recapture* method to address the problem. This method has been applied successfully in the wildlife research domain to estimate the population size and adapted to the task of measuring software quality and managing the software inspection process.

Capture-Recapture (CR) is a statistical method that is used by repeatedly trapping (or capturing) a fixed number of animals, marking them, and releasing them back into the population. If the same animal is trapped during subsequent trapping occasions, it said to have been recaptured. The size of a population is estimated using: 1) the total number of unique animals captured across all trapping occasions and 2) the number of animals that were re-captured. A higher percentage of recaptures indicates a smaller population [96].

Defects and inspection events in software engineering are analogous to the animals and trapping occasions in wildlife research. Therefore, using the same principle, the CR method can be used during the software inspection process to estimate the number of faults in an artifact. In an inspection, each inspector finds (or *captures*) a subset of total faults present in an artifact. When an inspector finds a fault that was captured by another inspector, it has been *recaptured*. Therefore, CR in software inspections uses the overlap in the faults found by multiple inspectors to estimate the number of faults in the artifact. The difference between the estimated number of faults and the number of faults that have been found provides an estimate of how many faults remain.

To investigate the use of the CR method in software inspections, and its usefulness for software project managers, this dissertation has a following important thrust (which will be more fully discussed in Chapter V):

- Empirically validate the use of the capture-recapture method for helping software developers and project managers monitor the quality of software artifacts and make correct re-inspection decisions.

1.3 Research Hypotheses

This dissertation hypothesizes that the requirement error taxonomy can be developed and the requirement error taxonomy will be an effective quality improvement approach as stated below:

“A taxonomy of requirement errors can be developed and will be an effective software approach for improving software quality”

This dissertation also hypothesizes that the capture-recapture method is useful for measuring the quality of software artifacts and managing the software inspection process as stated:

“The capture-recapture models can support software inspections by estimating the post-inspection defects and predicting the need of re-inspection of software artifacts”.

1.4 Framework of Research Activities

This dissertation has three major research thrusts (as described in Sections 1.1 and 1.2), The research activities involved in this dissertation can be classified into three different phases, and are described as:

- Development of the Requirement Error Taxonomy,

- Empirical Validation of the Requirement Error Taxonomy, and
- Validating the usefulness of the Capture-Recapture method for helping project managers determine the quality of software product under construction and manage the software inspection process.

1.5 Organization

The remainder of this dissertation is organized as follows. Chapter II reviews the related literature from the fields of Software Engineering, Cognitive Psychology, and Wildlife Research. Chapter III introduces the proposed research approach for solving the problems described in this chapter, details the process of developing the requirement error taxonomy, presents the requirement error taxonomy, and outlines the empirical approach used in this dissertation. Chapter IV and Chapter V detail the experiment designs, data analysis, and results from the empirical studies conducted to validate the Requirement Error Taxonomy and the Capture-Recapture method. Chapter VI and Chapter VII discuss the contributions of this dissertation to researchers and practitioners, state the conclusion, and present future research directions.

CHAPTER II

LITERATURE REVIEW

The idea of using the sources of faults to improve software quality is not novel. Prior research has focused on fault origin to develop quality improvement approaches. While many of these approaches are useful, they have two main shortcomings. First, they do not typically define a formal process for finding and fixing errors. Second, they may be incomplete because they were developed based on a sample of observed faults rather than on a strong cognitive theory that provides comprehensive insight into human mistakes. Section 2.1 discusses three major research efforts that have exploited the source of faults. Section 2.2 provides an overview of how cognitive psychology research can help with the task of understanding software development errors. Section 2.3 then discusses the results from a feasibility study performed during my Master's Thesis, which motivated this dissertation. Regarding the task of investigating the use of capture-recapture in software inspections, Section 2.4 details the basic principles and assumptions involved in the use of the capture-recapture models to estimate wildlife populations and how successfully those models can be used in software engineering to estimate the abundance of faults. Section 2.4 also discusses previous research in the use of capture-recapture in software engineering, highlighting the areas that need further research.

2.1 *Research on the Sources of Faults*

Identification of the source of faults is referred to by various names (Root Cause Analysis, Defect Cause Analysis, Software Failure Analysis, and Software Bug Analysis). The goal of this line of research is to identify systematic problems in a development organization as a basis for process improvement (e.g., [20, 38, 42, 43, 54, 55, 56]). While each of these approaches uses a different process, a common theme is that they focus on finding the source of faults that are found late in the lifecycle. In Card's approach, faults are stored in a database and analyzed separately from the development process [20]. The Root Cause Analysis approach, by Leszak, et al., provides a set of multi-dimensional *triggers* to help developers characterize the fault source [51]. Grady, et al., define an approach where only a sample of the faults are analyzed with the goal of finding common sources for classes of faults (e.g. User Interface faults) [38]. The approach by, Jacobs, et al., makes use of accumulated expert knowledge rather than conducting a detailed analysis of each fault [42]. My work builds upon these findings by emphasizing faults that can be found early in the lifecycle (i.e., during the requirements phase). Each of these approaches and the list of errors resulting from them are discussed in more detail as part of the systematic literature review in Chapter III.

Similarly, the Orthogonal Defect Classification (ODC) is an in-process method in which developers classify faults using a predefined taxonomy. Then, the developers identify a *trigger* that revealed the failure (not necessarily the cause of fault insertion). Because the triggers explain the actions that revealed the failure at the code level, ODC is more objective than identifying the cause of fault insertion. This approach has been

shown to provide useful feedback to developers [21]. My work builds on this concept by helping developers understand not only what caused a fault to be revealed, but more interestingly what caused the fault to be inserted in the first place.

The third approach, the *Error Abstraction* by Lanubile, et al., helps developers examine the faults detected during an inspection to determine their underlying cause, i.e. the error. The process consists of three major steps. First, the artifact is inspected to find as many faults as possible. Second, the inspectors are given instructions on how to identify the underlying errors that led to faults or groups of faults. This step is called error abstraction. Finally, the inspectors re-inspect the requirements looking for any additional faults that could be the result of the errors identified in the second step, which were not found during the first inspection. The initial work showed promising results for using the error abstraction approach, but this line of research was not extended [49]. One of the shortcomings of this approach was that the error abstraction guidance was not very concrete and relied heavily on the expertise of the individual inspector. This dissertation work extends the error abstraction approach by providing a formal error taxonomy, which includes research from cognitive psychology, to support developers during the error abstraction and re-inspection process.

2.2 *A Cognitive Psychology Perspective on Errors*

While psychological study of human errors began during the 1920's [67], two large accidents (the 1994 Bhopal pesticide accident and the 1996 Chernobyl nuclear power plant accident) spurred renewed interest in the field (e.g. [20]). Systematic human

error models were built on basic theoretical research in human cognition, especially from an information processing approach. It quickly became apparent that errors were not the result of irrational behavior, but rather resulted from normal psychological processes gone awry. Two approaches for studying human error have emerged. The first approach focuses on an individual's actions and the psychological processes that resulted in error. The second approach is focused on errors at the system level (e.g. problems in communication, training, and safety mechanisms within an organization). Each approach has contributed an important perspective on the origins and types of human error and provided methods to reduce human error.

Reason [67] introduced the Generic Error-Modeling System (GEMS) to explain errors made by individuals as they work. Errors occur at three different levels: 1) *Skill-based errors* arise when routine actions are erroneously carried out in a familiar environment; 2) *Rule-based errors* arise when a familiar if-then rule is erroneously applied in an inappropriate situation; and 3) *Knowledge-based errors* occur when reasoning and planning solutions to novel problems or situations. In software engineering, skill-based errors may appear either during coding (e.g., a programmer forgot to include a buffer-overflow check even though he intended to) or during the requirements or design (e.g., by omitting important details when documenting a familiar environment). Rule-based errors are more likely to appear in the design phase, when a designer may select a familiar design pattern even though it is not appropriate for the current system. Finally, a knowledge-based error may occur when the software engineer

fails to understand the unique aspects of a new domain and as a result produces an incomplete or incorrect set of software requirements.

A key assumption of the GEMS approach is that, when confronted with a problem, people tend to find a prepackaged solution at the rule-based level before resorting to the far more effortful knowledge-based level, even when the latter is demanded at the outset [67]. The implication of this tendency is that software engineers are likely to employ familiar requirements engineering approaches even when these approaches are inappropriate and lead to faults. The GEMS approach defines several skill-based, rule-based, and knowledge-based errors resulting from factors such as attention, information overload, and problems with human reasoning about if-then rules in familiar and novel situations.

At the organizational level, Rasmussen [65-66], employed a similar skill-rule-knowledge framework. Rasmussen defines: 1) *skill* as the actions which come natural as a result of practice and require no conscious checking; 2) *rule* as the type of behavior which follows a standard sequence that has been developed through experience; and 3) *knowledge* as the behavior involved in reacting to novel situations for which no standard working method is available. Rasmussen focused on the human information processing and corresponding knowledge states during the decision-making process. By focusing at the level of the entire system that produced the error, rather than on the individual who made the obvious error, Rasmussen's approach has helped to foster error-tolerant systems. This approach has been particularly valuable in industrial situations, where accidents may have tragically large costs. Rasmussen's model can provide insights into

the underlying cognitive failure responsible for errors made by software developers. For example, errors can occur while developers are trying to retrieve information about the software system (i.e., information errors), or using the available information to choose wrong development methods for achieving the intended goal (i.e., strategy errors), or incorrect execution of the selected methods (i.e., procedure errors).

Both individual and systems-level accounts of human errors have been successfully used to explain errors in a broad range of disciplines and to introduce new safety mechanisms that have reduced the probability of errors. The benefits have been seen in the investigation of medical errors, aviation errors, and the Chernobyl nuclear power plant explosion. Even with these successes, the understanding of how to apply human error research to software engineering is not obvious. Much of the understanding of human error has come from case studies of accidents. Errors in software engineering processes are somewhat different than the examples previously discussed. Therefore, the models of human error will have to be adapted to software errors. A better understanding of the origin of faults (errors), and of the means to reduce faults, will contribute positively to software engineering.

2.3 Related Work on the Use of Error Information to Improve Software Quality

Before embarking on the approach to integrate cognitive psychology research into the software quality process, it was prudent to perform a feasibility study to determine whether human error research could provide benefits to software quality. To evaluate the feasibility, I developed and validated the first version of requirements error taxonomy.

During my Master's Thesis, I performed an ad-hoc review of the software engineering literature and the cognitive psychology literature to create an initial version of the requirements error taxonomy including errors from both sources [86].

To evaluate the usefulness of this initial version of the taxonomy, I then conducted a repeated-measures quasi-experiment in a senior-level full-year capstone project where the students developed a real system for a real client. In this study, the subjects performed a traditional inspection of the requirements document they created for their project. Then they examined the list of faults found during that inspection to determine the underlying errors. They were then taught the requirements error taxonomy and asked to classify the errors using the taxonomy. After the students were aware of the errors they had committed, they were asked to use that information to perform a re-inspection to find additional faults. The results of the study indicated that qualitatively the subjects found the error taxonomy both easy to use and effective. In addition, the quantitative results showed that by using the error taxonomy, the subjects were able to find significantly more defects during the re-inspection. Finally, as an indication of the usefulness of the human error information, ten of the twelve subjects found errors related to the human errors from cognitive psychology. The percentage of errors that could be traced to human errors ranged from 10%-20% of the total errors reported [86].

The promising results from the feasibility study in my Master's Thesis motivated:

- 1) the need of a more systematic approach to gather the error information from the software engineering and psychology literature to refine the requirement error taxonomy,
- and 2) the need to better understand the results through extensive empirical validation of

the requirement error taxonomy. These needs were addressed by this dissertation by: 1) performing a systematic literature review to develop the requirement error taxonomy (as described in Chapter III), and 2) performing an extensive empirical validation of the requirement error taxonomy (as described in Chapter IV).

2.4 Use of Capture-Recapture in Software Inspections

This dissertation investigates the use of the capture-recapture method in software inspections to support estimates of the remaining defects in a software artifact. Section 2.4.1 describes the basic principles and assumptions of the capture-recapture models in wildlife research, their application to software inspections, and the different capture-recapture models and estimators that are described in the literature. Section 2.4.2 discusses the background literature that motivated the need for research into the application of the capture-recapture method to software inspections.

2.4.1 Capture-Recapture Models and Estimators

While software engineering researchers can borrow the theory of capture-recapture models from biology and wildlife research, the use of capture-recapture models in biology and wildlife research makes certain assumptions that are not always met in the context of software inspections [17]. The assumptions made by capture-recapture models include a closed population (i.e., animals cannot enter or leave the population), equal capture probability (i.e., the probability of capturing animals at each trapping occasion is

same and the probability of each animal being captured is the same), and that marks are not lost during capture and recapture (i.e., marked animals remain marked) [96].

The application of capture-recapture models to software inspections meets the assumption of a closed population (i.e., all inspectors review the same artifact and detect defects from the same pool of defects) and the assumption that marks are not lost during capture and recapture (i.e., defects are recorded on a defect list and easily maintained). However, the assumption about *equal capture probability* is not met for software inspections because some defects are easier to detect than others. In addition, capture probabilities can vary among different inspectors due to different defect detection capabilities (based on education, training, background, or innate ability) [17].

To accommodate these assumptions, the capture-recapture models for software inspections are built around two sources of variations: *Inspector Defect Capability* (inspectors differ in their ability to detect defects) and *Defect Detection Difficulty* (different defects will have different probability of being detected). Different CR models

Table 2.1

CAPTURE-RECAPTURE MODELS [17]

Model	Variation Source
M_o	Inspector Ability – Constant; Defect Detection Likelihood – Constant.
M_t	Inspector Ability – Variable; Defect Detection Likelihood – Constant.
M_h	Inspector Ability – Constant; Defect Detection Likelihood – Variable.
M_{th}	Inspector Ability – Variable; Defect Detection Likelihood – Variable.

have been developed based on whether these two variables are assumed to be constant or not. Based on these sources of variations, the four capture-recapture models shown in Table 2.1 have been used in previous software inspection studies.

Each capture-recapture model in Table 2.1 has a set of estimators, which use different statistical approaches to produce the estimates. Table 2.2 shows the various estimators used in this study and which model they correspond to. Some of these estimators have been evaluated in previous software inspection studies and some are new estimators from biology that have not previously been applied to software inspections (marked with an *). The details of the estimators and how they calculate the defect estimates can be found in the provided references. All of the capture-recapture models

Table 2.2

CAPTURE-RECAPTURE ESTIMATORS

Models	Estimators
M₀	Unconditional Maximum Likelihood Estimator (M ₀ -UMLE)
	*Conditional Maximum Likelihood Estimator (M ₀ -CMLE)
	*Estimating Equations (M ₀ -EE)
M_t	Unconditional Maximum Likelihood Estimator (M _t -UMLE)
	*Conditional Maximum Likelihood Estimator (M _t -CMLE)
	*Estimating Equations (M _t -EE)
	Chaos Estimator (M _t -Ch)
M_h	Jackknife Estimator (M _h -JK)
	*Sample Coverage (M _h -SC)
	*Estimating Equations (M _h -EE)
	Chaos Estimators (M _h -Ch)
M_{th}	*Sample Coverage (M _{th} -SC)
	*Estimating Equations (M _{th} -EE)

use the same inspection data organized in the same way: a simple matrix in which each defect is represented by a row and each inspector is represented by a column (Figure 2.1).

An entry in the matrix is a 1 if the particular defect represented by the row was found by the inspector represented by the column and a 0 otherwise. While each estimator makes different assumptions about the data, using this matrix all of the statistics required by each estimator can be calculated [96]

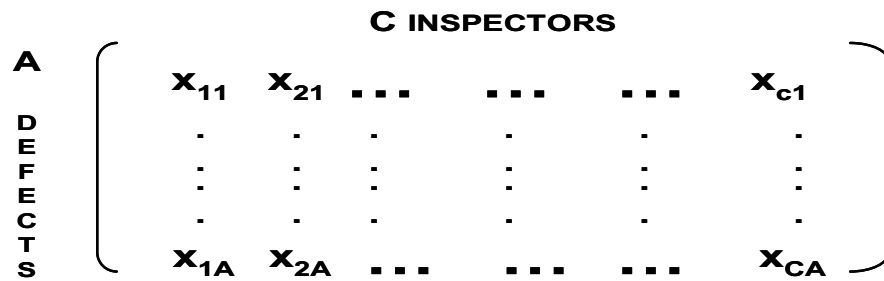


Figure 2.1

CR Data Input Matrix

2.4.2 Empirical Studies on Capture-Recapture in Software Inspections

Eick, et al., introduced the theory of capture-recapture models and estimators for software inspections by applying those concepts to defect data from AT&T [26]. They used the maximum likelihood estimator (MLE) for the M_t model and compared the resulting estimate with the inspector's subjective estimates of the remaining defects. The major result of this study was that the capture-recapture estimates were consistent with the subjective opinion of inspectors. Also, Eick, et al., recommended that an artifact should be re-inspected when the estimated number of remaining defects is greater than

20% of the total [26, 27]. This recommendation is still used by capture-recapture studies for evaluating estimators.

Later, in the same AT&T environment, Weil and Votta studied the performance of the M_t -MLE and M_h -JK (Jackknife) estimators to understand what occurs when the assumptions of the models are violated. The results from this study show that while both estimators produced inaccurate estimates, the MLE estimator was somewhat better because it underestimated the number of defects while the Jackknife estimator severely overestimated the number of defects. They proposed a method for improving the accuracy of these estimators by organizing faults into smaller groups prior to performing the estimation. Also, the variability of the Jackknife estimator was greater than the MLE estimator. The results showed that grouping improves the accuracy of the M_t -MLE estimator but not the M_h -JK estimator [95].

Building on these results, Briand, et. al., evaluated a series of capture-recapture models using data generated from the inspection of a set of seeded artifact by software professionals from NASA. This study also investigated the effect that the number of reviewers had on the performance of the estimators by varying the input size from one to six inspectors. A finding from this study, which was consistent with earlier studies, was that the models generally underestimate. The results also show that a minimum of four reviewers are needed to calculate satisfactory estimates. Also, they recommended the use of the Jackknife estimator when using data from four or more reviewers [17]. Conversely, other studies that examined capture-recapture using two inspectors concluded that the estimates are unreliable when the number of inspectors is small, and the Jackknife

estimator severely underestimates in such conditions [28, 83]. The studies in software engineering have been limited to relatively small data sets with a small number of inspectors and defects. Since the capture-recapture models work based on the amount of overlap in the defects detected by different inspectors, it is unclear “*What effect a large number of inspectors and defects will have on the performance of the capture-recapture models*”? This dissertation investigates this question in more detail.

Emam, et al., also evaluated CR estimators for two inspectors using artifacts with seeded defects and analyzed their ability to accurately determine the need for re-inspection. The results showed that M_h -JK and M_t -Ch are the best estimators and that not all estimators helped in making correct decisions on re-inspections [28]. Emam, et al., also advocated the use of the inspectors’ subjective opinions along with the CR estimates when used during real development (i.e., when the actual defects are unknown rather than seeded) [29]. Conversely, another study showed that the subjective estimates are significantly less accurate than the CR estimates [83]. Analysis of these empirical studies over ten years (1992-2002) of the CR research in software inspections, asserts that the results regarding the use of CR are derived from studies conducted on the artifacts with seeded defects [62]. There is little evidence to support the efficacy of using CR models in real software development (with an unknown number of naturally occurring defects). Moreover, the empirical studies until now have focused on a few selected estimators for each CR model type.

A more detailed analysis of all the available estimators for each CR model is needed. Therefore, this dissertation investigates these following open research issues in

order to build a body of knowledge that will guide the project managers and software developers on how to use the capture-recapture method in their organizations:

- Investigate the effect of the *number of inspectors* and the *number of faults* on the estimates produced by capture-recapture models and estimators; and
- Evaluate the use of capture-recapture models and estimators for estimating the abundance of defects using artifacts with an unknown number of naturally occurring defects.

CHAPTER III

RESEARCH APPROACH

This chapter describes the research approach used to develop the requirement error taxonomy. Section 3.1 describes the systematic literature review process as a research method for developing the requirement error taxonomy. Section 3.2 details the steps involved in the review process. Section 3.3 presents the results of the review, Section 3.4 describes the requirement error taxonomy. Section 3.5 provides an overview of the empirical approach to validate the error taxonomy. Finally, Section 3.6 provides an overview of the empirical investigation into the capture-recapture method.

3.1 Research Method for Developing Requirement Error Taxonomy

As a first step, I performed a systematic literature review to identify and classify the errors identified by other software quality researchers. A *systematic literature review* is a formalized, repeatable process by which researchers systematically search a body of literature to document the state of knowledge on a particular subject. The benefits of performing a systematic review, as opposed to using the more common *ad hoc* approach, is that it provides the researchers with more confidence that they have located as much relevant information as possible. This approach is commonly used in other fields to document high-level conclusions that can be drawn from a series of detailed studies [44].

To be effective, a systematic literature review must be driven by an overall goal.

In this review, a high level goal was to:

Identify and classify types of requirement errors into a taxonomy to support the prevention and detection of errors.

3.2 Description of the Systematic Review Process

Following published guidelines [44], the systematic review process included the following steps:

- Formulate a review protocol,
- Conduct the review (identify primary studies, evaluate those studies, extract data, and synthesize data to produce a concrete result),
- Analyze the results,
- Report the results, and
- Discuss the findings.

The review protocol specified the questions to be addressed, the databases to be searched and the methods to be used to identify, assemble, and assess the evidence. To reduce researcher bias, the protocol, described in the remainder of this section, was developed by me, reviewed by my advisor and then finalized through discussion, review, and iteration with the empirical software engineering research group. An overview of the review protocol is provided here. The complete, detailed description has already been published [88, 94].

3.2.1 Research Questions

The main goal of this systematic review was to identify and classify different types of requirement errors. To properly focus the review, a set of research questions was needed. With the underlying goal of providing support to the software quality improvement process, the high-level question addressed by this review was:

What types of requirements errors can be identified from the literature and how can they be classified?

Motivated by the background research and to complement the types of errors identified in the software engineering literature, this review makes use of cognitive psychology research into understanding human errors.

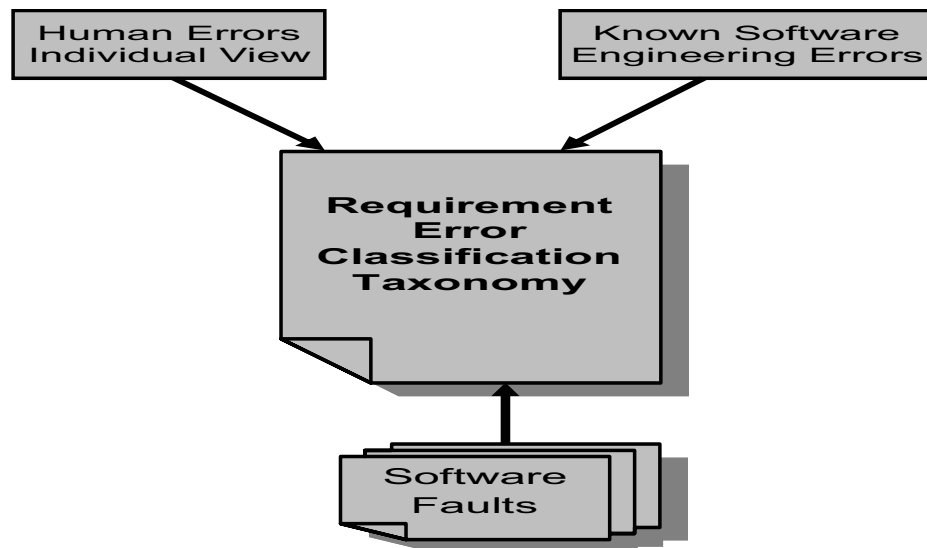


Figure 3.1

Types of Literature Searched to Identify the Requirement Errors

Figure 3.1 shows that the results of a literature search in software engineering about faults and their causes are combined with the results of a literature search of the more general psychological accounts of human errors to develop the requirement error taxonomy.

Table 3.1

RESEARCH QUESTIONS AND MOTIVATIONS

Research Question	Motivation
1. Is there any evidence that using error information can improve software quality? <i>1.1. Are there any processes or methods reported in literature that use error information to improve software quality?</i> <i>1.2. Do any of these processes address the limitations and gaps identified in the earlier quality improvement approaches?</i>	Assess the usefulness of errors in existing approaches; identify shortcomings the current approaches and avenues for improvement
2. What types of requirement errors have been identified in the software engineering literature? <i>2.1. What types of errors can occur during the requirement stage?</i> <i>2.2. What errors can occur in other phases of the software lifecycle that are related to requirement errors?</i> <i>2.3. What requirement errors can be identified by analyzing the source of actual faults?</i>	Identify types of errors in the software engineering literature as an input to an error classification taxonomy
3. Is there any research from human cognition or psychology that can propose requirement errors <i>3.1. What information can be found about human errors and their classification?</i> <i>3.2. Which of the human errors identified in Question 3.1 can have corresponding errors in software requirements?</i>	Investigate the contribution of human errors from the fields of human cognition and psychology
4. <i>How can the information gathered in response to Questions 1-3 be organized into an error taxonomy</i>	Organize the error information into a taxonomy

The high-level research question was decomposed into the specific research questions and sub-questions shown in Table 3.1, which guided the literature review.

The first question required searching the software engineering literature to identify any quality improvement approaches that focus on errors. These approaches were analyzed to identify any shortcomings and record information about errors. The second question also required searching the software engineering literature with the purpose of explicitly identifying requirement errors and errors from other lifecycle phases (i.e., the design or coding phase) that are related to requirement errors. In addition, this question also required analysis of faults to identify the underlying errors. As a result, this question identified a list of requirement errors from the software engineering literature.

To answer the third question, we searched the human error literature from the human cognition and psychology fields to identify other types of errors that may occur while developing a requirement artifact. This question involved analyzing the models of human reasoning, planning, and problem solving and their fallibilities to identify errors that can occur during the requirements development stage. Finally, the fourth research question, which is really a meta-question using the information from questions 1-3, combines the errors identified from the software engineering literature and the human cognition and psychology literature into a requirement error taxonomy.

The initial list of source databases was developed using a detailed source selection criteria. The database searches resulted in an extensive list of potential papers. To ensure that all papers included in the review were clearly related to the research questions, detailed inclusion and exclusion criteria were defined.

Table 3.2 shows the inclusion and exclusion criteria for this review. The inclusion criterion is specific to each research question, while the exclusion criterion is common for all questions. Using this process, the initial search returned over 25,000 papers, which were narrowed down to 7838 papers based on their titles, then to 482 papers based on their abstracts and keywords. Then, these 482 papers were read to select the final list of 149 papers based on the inclusion and exclusion criteria. Of these 149 papers, 108 were published in thirteen leading journals and six conferences in software engineering, and 41 were published in nine psychology journals. The distribution of the selected papers is shown in Table 3.3.

Table 3.2

INCLUSION AND EXCLUSION CRITERIA

RQ	Inclusion Criteria	Exclusion Criteria
1	<ul style="list-style-type: none"> • Papers that focus on analyzing/using the errors (source of faults) for improving software quality, • Empirical studies (qualitative or quantitative) of using the error information in software lifecycle 	<ul style="list-style-type: none"> • Papers that are based only on expert opinion • Short-papers, introductions to special issues, tutorials, and mini-tracks • Studies not related to any of the research questions • Preliminary conference versions of included journal papers • Studies not in English • Studies whose findings are unclear and ambiguous (i.e., results are not supported by any evidence)
2	<ul style="list-style-type: none"> • Papers that talk about errors, mistakes or problems in the software development process and requirements in particular, • Papers about error, fault, or defect classifications, • Empirical studies (qualitative or quantitative) on the cause of software development defects. 	
3	<ul style="list-style-type: none"> • Papers from human cognition and psychology about human thought process, planning, or problem solving, • Empirical studies on human errors, • Papers that survey or describe the human error classifications 	

Table 3.3

PAPER DISTRIBUTION

Source	Count	%
IEEE Computer	19	12.8%
Journal of System and Software	13	8.7%
Journal of Accident Analysis and Prevention	11	7.4%
ACM Transactions on Software Engineering	11	7.4%
Communications of the ACM	8	5.4%
IBM Systems Journal	8	5.4%
IEEE Transaction in Software Engineering	8	5.6%
ACM Transactions on Computer-Human Interaction	6	4%
Applied Psychology: An International Review	6	4%
SEI Technical Report Website	5	3.4%
Journal of Information and Software Technology	5	3.4%
IEEE Int'l Symposium on Software Reliability Engineering	4	2.7%
Journal of Ergonomics	4	2.7%
IEEE Trans. on Systems, Man, and Cybernetics (A): Systems & Humans	4	2.7%
IEEE International Symposium on Empirical Software Engineering	4	2.7%
Requirements Engineering Journal	4	2.7%
International Conference on Software Engineering	3	2%
Journal of Computers in Human Behavior	3	2%
Empirical Software Engineering: An International Journal	3	2%
The International Journal on Aviation Psychology	2	1.3%
IEEE Annual Human Factors Meeting	2	1.3%
International Journal of Human-Computer Interaction	2	1.3%
Software Process: Improvement and Practice	2	1.3%
Journal of Software Testing, Verification and Reliability	1	0.6%
Journal of Reliability Engineering and System Safety	1	0.6%
IEEE International Software Metrics Symposium	1	0.6%
Journal of Information and Management	1	0.6%
Software Quality Journal	1	0.6%
High Consequence System Surety Conference	1	0.6%
Crosstalk: The Journal of Defense Software Engineering	1	0.6%
Journal of IEEE Computer and Control Engineering	1	0.6%
Total	149	100%

3.2.2 *Data Extraction and Synthesis*

Data extraction forms were used to ensure consistent and accurate extraction of the important information from each paper related to the research questions. In developing the data extraction forms, it was determined that some of the information was needed regardless of the research question, while other information was specific to each research focus. Consistent with the process followed in previous systematic reviews (e.g. [45]), I reviewed all papers and extracted data. Then, my advisor independently reviewed and extracted data from a sample of the papers. We then compared the data extracted by each reviewer for consistency. We found that we had consistently extracted information from the sample of papers, suggesting that my advisor did not need to review the remainder of papers in detail and that the information extracted by me was sufficient. The data extracted from all papers was synthesized to answer each research question described in Section 3.3.

3.3 *Reporting the Review: Answering Research Questions*

Question 1: Is there any evidence that using error information can improve software quality?

The idea of using the source of faults to improve software quality is not novel. Other researchers have used this information in different ways with varying levels of success. A review of the different methods indicates that knowledge of the source of faults is useful for process improvement, defect prevention by helping developers learn from their mistakes, and detection of defects during inspections. A major drawback of these approaches is that they typically do not provide a formal process to assist

developers in finding and fixing errors. In fact, only the approach by Lanubile, et al., provided any systematic way to use error information to improve the quality of a requirements document [53]. Another drawback to these methods is that they rely only on a sample of faults to identify errors, therefore potentially overlooking some errors. While these approaches have positive aspects and describe software errors (as discussed in the answers to 1.1), they also have limitations (as discussed in the answers to 1.2). A total of thirteen papers addressed this question and its related sub-questions.

Question 1.1: Are there any processes or methods reported in literature that use error information to improve software quality?

The review identified nine methods that stress the use of error information during software development. These methods are described in this section along with their use in software organizations, and the types of errors they helped identify.

- The *defect causal analysis* approach is a team-based quality improvement technique used to analyze samples of previous faults to determine their cause (the error) in order to suggest software process changes and prevent future faults. Empirical findings show that the use of the defect causal analysis method at IBM and at Computer Sciences Corporation (CSC) resulted in a fifty percent decrease in defects over each two year period studied. Based on the results from these studies, Card suggest that all causes of software faults fall into one of four categories: a) methods (incomplete, ambiguous, wrong, or enforced), b) tools and environment (clumsy, unreliable, or defective), c) people (who lack adequate training or understanding), and d) input and requirements (incomplete, ambiguous, or defective) [20].
- The *software failure analysis* approach is a team-based quality approach. The goal of this process is to improve the software development and maintenance process. Using this approach, developers analyze a representative sample of defect data to understand the causes of particular classes of defects (i.e., specification, user-interface etc). This process was used by engineers at Hewlett-Packard to understand the cause of user-interface defects and develop better user interface design guidelines. During their next year-long project, using these guidelines, the user-interface defects found during test decreased fifty percent. All the causes of user-interface defects were classified into four

different categories: guidelines not followed, lack of feedback or guidelines, different perspectives, or oops! (forgotten) [38].

- The *defect causal analysis (using experts)* approach accumulates expert knowledge to substitute for an in-depth, per-object defect causal analysis. The goal is to identify the causes of faults found during normal development, late in the development cycle and after deployment. Based on the results of a study, Jacobs, et al., describe the causes of software defects e.g., inadequate unjustified trust, inadequate communication, unclear responsibilities, no change control authority, usage of different implementations etc. The study also showed that the largest number of defects were the result of communication problems [42].
- The *defect prevention process* uses causal analysis to determine the source of a fault and to suggest preventive actions. Study results show that the process helps in preventing commonly occurring errors, provides significant reductions in defect rates, leads to less test effort, and results in higher customer satisfaction. Based on the analysis of different software products over six years, Mays, et al., classified defect causes as oversight causes (e.g., developer overlooked something, or something was not considered thoroughly), education causes (developer did not understand some aspect), communication causes (e.g., something was not communicated), or transcription causes (developer knew what to do but simply made a mistake) [55].
- The *software bug analysis* process identifies the source of bugs. It also contains countermeasures (with implementation guidance) for preventing bugs. To assess the usefulness of this approach, a sample of twenty-eight bugs found during the debug and test phases of authorization terminal software development were analyzed by group leaders, designers, and third party designers to determine their cause. A total of twenty three different causes were identified, and more than half of these causes were related to designers carelessness [56].
- The *root cause analysis* method introduced the concept of a multi-dimensional *defect trigger* to help developers determine the root cause of a fault (the error) in order to identify areas for process improvement. Leszak, et al., conducted a case-study of the defect modification requests during the development of a transmission network element product. They found the root causes of defects to include: communication problems or lack of awareness of the need for communication, lack of domain/system/tools/process knowledge, lack of change coordination, and individual mistakes [51].

- The *error abstraction process* analyzes groups of related faults to determine their cause (i.e. the error). This error information is then used to find other related faults in a software artifact [49].
- The *defect based software process improvement* analyzes faults through attribute focusing to provide insight into their potential causes and makes suggestions that can help a team adjust their process in real time [20, 54].
- The *goal-oriented process improvement* methodology also uses defect causal analysis for tailoring the software process to address specific project goals in a specific environment. Basili and Rombach describe an error scheme that classifies the cause of software faults into different problem domains. These problem domains include: application area (i.e., misunderstanding of application or problem domain), methodology to be used (not knowing, misunderstanding, or misuse of problem solution processes), and the environment of the software to be developed (misunderstanding or misuse of the hardware or software of a given project) [7].
- *Total quality management* is a philosophy for achieving long-term success by linking quality with customer satisfaction. Key elements of this philosophy include total customer satisfaction, continuous process improvement, a focus on the human side of quality, and continuous improvement for all quality parameters. It involves identifying and evaluating various probable causes (errors) and testing the effects of change before making it [43]. This approach is based on the *defect prevention process* described earlier.

Each of the above methods uses error information to improve software quality.

Each has shown some benefits and identified some important types of errors. This information served as an input to the requirement error taxonomy.

Question 1.2: *Do any of these processes address the limitations and gaps identified in previous quality improvement approaches?*

Each of the methods described in the answer to *Question 1.1* does address many of the limitations, but, they still have some additional limitations. The limitations common to the different methods are summarized in Table 3.4. While these methods do analyze the source of faults, they only use a small sample of faults, potentially overlooking some errors. As a result, the error categories are generic and appear to be

incomplete. Moreover, some methods only describe the cause categories (and not actual errors), while others provide only a few examples for each category. None of the methods provides a list of all the errors that may occur during the requirements phase.

Another common problem is that while many methods do emphasize the importance of *human errors* and provides a few examples (e.g., errors due to designer's carelessness and transcription errors), they do not go far enough. These approaches lack a strong cognitive theory to describe a more comprehensive list of errors. Finally, no formal process is available that can guide developers in identifying the errors and resulting faults when they occur. The inability of the existing methods to overcome these limitations motivates the need for development of a more complete requirement error taxonomy.

Table 3.4

LIMITATIONS OF METHODS IN QUESTION 1.1

Limitations
Requires extensive documentation of problem reports and inspection results; and over-reliance on historical data
Cost of performing the causal analysis through implementing actions ranges from 0.5 to 1.5 percent of the software budget; and requires a startup investment.
It is cost-intensive, people-intensive and useful for analyzing only a small sample of faults, or group of related faults
Requires experienced developers, extensive meetings among experts, and interviewing the actual developers to analyze the probable causes of faults
Results in a large number of actions, and has to applied over a period of time to test the suggested actions/improvements to software process
It can only detect the process inadequacy/ process improvement actions and not reveal the actual error
It analyzes the faults found late in the software lifecycle.
Does not guide the inspectors, and relies heavily on the creativity of inspectors in abstracting errors from faults during software inspections

Question 2: *What types of requirement errors have been identified in the software engineering literature?*

The identification of requirement errors supports future research into software quality. To obtain an initial list of errors, the software engineering literature was reviewed to extract any errors that had been described by other researchers. This research question is addressed in detail by sub-questions 2.1 and 2.2. A total of fifty five papers were analyzed to identify the types of requirement errors in the software engineering literature, thirty to address question 2.1 and twenty four to address question 2.2.

Question 2: *What types of errors can occur during the requirement stage?*

The review uncovered a number of candidates for requirement errors. Table 3.5 lists potential requirement errors along with the relevant references.

The first row of Table 3.5 contains information from six studies that analyzed faults found during software development using different causal analysis methods (as

Table 3.5

SOURCES USED TO IDENTIFY REQUIREMENT ERRORS IN THE LITERATURE

Sources of Errors	References
Root causes, cause categories, bug causes, defect-fault causes	[11, 20, 38, 42, 55-56]
Requirement engineering problem classification	[13, 18, 39, 74, 79-81]
Empirical studies on root causes of troubled projects or errors	[6, 36, 71]
Influencing factors in req. stage and software development	[8, 22, 78, 97]
Causes of requirement traceability and requirement inconsistency	[24, 32, 60, 68]
Domain knowledge problems	[60]
Management problems	[24]
Situation Awareness / Decision making errors	[32]
Team errors	[68]
Other	[37, 52-53, 85]

described in Question 1.1) to identify requirement errors [11, 20, 38, 42, 55-56]. Examples of the errors identified by these studies include: lapses in communications among developers, inadequate training, not considering all the ways a system can be used, and misunderstanding certain aspect of the system functionality.

The second source of errors includes empirical studies that classify requirement error problems experienced by developers, studies that classify difficulties in determining requirements, and other case studies that trace requirement errors to the problems faced during the requirements engineering process [13, 18, 39, 74, 79-81]. Examples of the errors from these studies include: lack of user participation, inadequate skills and resources, complexity of application, undefined requirement process, and cognitive biases.

The third source of errors include studies that describe the root causes (some of whom were those belonging to requirement stage) of troubled IT projects [6, 36, 71]. Examples of the errors from these studies include: unrealistic business plan, use of immature technology, not involving users at all stages, lack of experienced or capable management, and improper work environment.

The fourth source of errors is findings from an empirical study of thirteen software companies that identified thirty-two potential problem factors and other similar empirical studies that identified requirement problem factors affecting software reliability [8, 22, 78, 97]. Examples of the errors from these studies include: hostile team relationships, schedule pressure, human nature (mistakes and omissions), and flawed analysis method.

The fifth source of errors is requirement traceability and requirement inconsistencies, errors due to domain knowledge, requirement management errors, and errors committed among team members during software development [24, 32, 60, 68]. Examples of the errors from these studies include: poor planning, insufficiently formalized change requests, impact analysis not systematically achieved, misunderstandings due to working with different systems, and team errors.

Finally, Table 3.5 contains *other errors* that include the root causes of safety-related software errors, and the contribution of human error from social and organizational literature [37, 52-53, 85]. Examples of the errors from these studies include misunderstanding of assumptions and interface requirements, misunderstanding of dependencies among requirements, mistakes during application of well defined process, and attention or memory slips.

Question 2.2: *What errors can occur in other phases of the software lifecycle that are related to errors that can occur during the requirements phase?*

In addition to the errors found specifically in the requirements phase, the review also uncovered errors that occur during the design and coding phases. These errors can also occur during the requirements phase, including:

- *Missing Information:* Miscommunication between designers in different teams; lack of domain, system, or environmental knowledge; or misunderstandings caused by working simultaneously with several different software systems and domains [12, 38].
- *Slips in system design:* Misunderstanding of the current situation while forming a goal (resulting in an inappropriate choice of actions), insufficient specification of actions to follow for achieving goal (resulting in failure to complete the chosen actions), using an analogy to derive a sequence of actions from another similar situation (resulting in the choice of a sequence of actions

that is different from what was intended), or the sequence of actions is forgotten because of an interruption [58-59].

- *System programs*: Technological, organizational, historical, individual or other causes [30].
- *Cognitive breakdown*: Inattention, over attention, choosing the wrong plan due to information overload, wrong action, incorrect model of problem space, task complexity, or inappropriate level of detail in the task specification [46-47].

Question 2.3: *What requirement errors can be identified by analyzing the source of actual faults?*

In addition to identifying requirement errors directly reported in the literature, this review also identified a list of faults that could be traced back to their underlying error [3, 4, 5, 9, 10, 21, 35, 40, 63, 72]. For example, in the case where a requirements document is created by multiple sub-teams, a particular functionality is missing because each sub-team thought the other one would include it. The error that caused this problem is the lack of communication among the sub-teams. Another example is the case where two requirements are incomplete because they are both missing important information about the same aspect of system functionality. The error that caused this problem is the fact that developers did not completely understand that aspect of the system and it was manifested in multiple places. These errors were added to the list of errors that served as input to Question 4. These errors include:

- Misunderstanding or mistakes in resolving conflicts (e.g., there are unresolved requirements or incorrect requirements that were agreed on by all parties);
- Mistakes or misunderstandings in mapping inputs to outputs, input space to processes, or processes to output;
- Misunderstanding of some aspect of the overall functionality of the system;

- Mistakes while analyzing requirement use cases or different scenarios in which the system can be used; and
- Unresolved issues about complex system interfaces or unanticipated dependencies.

Question 3: *Is there any research from human cognition or psychology that can propose requirement errors?*

To address the fact that requirements engineering is a human-based activity and prone to errors, the review also examined the human cognition and psychology literature. The contributions of this literature to requirements errors are addressed by sub-questions 3.1 and 3.2. A total of thirty two papers that analyzed the human errors and their fallibilities were used to identify errors that may occur during the requirements phase:

Question 3.1: *What information can be found about human errors and their classification?*

The major types of human errors and classifications identified in human cognition and psychology include:

- *Reason's classification of mistakes, lapses, and slips:* Errors are classified as either *mistakes* (i.e., the wrong plan is chosen to accomplish a particular task), *lapses* (i.e., the correct plan is chosen, but a portion is forgotten during execution) or *slips* (i.e., the plan is correct and fully remembered, but during its execution something is done incorrectly) [19, 30, 59].
- *Rasmussen's skill, rule and knowledge based human error taxonomy:* Skill based slips and lapses are caused by mistakes while executing a task even though the correct task was chosen. Rule and knowledge based mistakes occur due to errors in intentions, including choosing the wrong plan, violating a rule, or making a mistake in an unfamiliar situation [19, 58, 37].
- *Reason's general error modeling system (GEMS):* a model of human error in terms of unsafe acts that can be intentional or unintentional. Unintentional acts include slips and lapses while intentional acts include mistakes and violations [19, 33, 58, 77].
- *Senders and Moray's classification of Phenomenological taxonomies, Cognitive taxonomies, and Deep Rooted Tendency taxonomies:* description of

the how, what and why concerns of an error, including omissions, substitutions, unnecessary repetitions, errors based on the stages of human information processing (e.g., perception, memory, attention), and errors based on biases [19, 77].

- *Swain and Guttman's classification of individual discrete actions*: Omission errors (something is left out), commission errors (something is done incorrectly), sequence errors (something is done out of order), timing errors (something is done too early or too late) [81].
- *Fitts and Jones control error taxonomy*: Based on a study of “pilot errors” that occur while operating aircraft controls. The errors include: substitution (choosing the wrong control), adjustment (moving the control to the wrong position), forgetting the control position, unintentional activation of the control, and inability to reach the control in time [33].
- *Cacciabue's taxonomy of erroneous behavior*: Includes system and personnel related causes. Errors are described in relation to execution, planning, interpretation, and observation along with the correlation between the cause and effects of erroneous behavior [19].
- *Galliers, Minocha, and Sutcliffe's taxonomy of influencing factors for occurrence of errors*: environmental conditions, management & organizational factors, task/domain factors, and user/personnel qualities including the slip and mistake types of errors described earlier [37].
- *Sutcliffe and Rugg's error categories*: operational description, cognitive causal categories, social and organizational causes, and design errors [79-81].
- *Norman's classification of human errors*: formation of intention, activation and triggering. Important errors include errors in classifying a situation, errors that result from ambiguous or incompletely specified intentions, slips from faulty activation of schemas, and errors due to fault triggering [57-59].
- *Human error identification (HEI) tool*: Describes the SHERPA tool that classifies errors as action, checking, retrieval, communication or selection [73, 77].
- *Human error reduction (HERA)*: Technique that analyzes and describes human errors in air the traffic control domain. HERA contains error taxonomies for five cognitive domains: perception & vigilance, working memory, long-term memory, judgment, planning & decision-making, and response execution [16, 41].

Question 3.2: Which of the human errors identified in Question 3.1 can have corresponding errors in software requirements?

Those errors that were relevant to requirements were included in the initial list of errors that served as input to Question 4 to make it more comprehensive. Examples of the translation of these errors into requirements errors are found in Table 3.6.

Table 3.6

REQUIREMENT ERRORS DRAWN FROM HUMAN ERRORS

Error	Description
Not understanding the domain	Misunderstandings due to the complex nature of the task; some properties of the problem space are not fully investigated; and, mistaken assumptions are made
Not understanding the specific application	Misunderstanding the order of events, the functional properties, the expression of end states, or goals
Poor execution of processes	Mistakes in applying the requirements engineering process, regardless of its adequacy; out of order steps; and lapses on the part of the people executing the process
Inadequate methods of achieving goals and objectives	System-specific information was omitted leading to the selection of the wrong technique or process; selection of a technique or process that, while successful on other projects, has not been fully investigated or understood in the current situation
Incorrectly translating requirements to written natural language	Lapses in organizing requirements; omission of necessary verification at critical points during the execution of an action; repetition of verification leading to the repetition or omission of steps
Other human cognition errors	Mistakes caused by adverse mental states, loss of situation awareness, lack of motivation, or task saturation; Mistakes caused by environmental conditions

Question 4: How can the information gathered in response to Questions 1-3 be organized into an error taxonomy?

Some of the errors were identified in more than one of the bodies of literature surveyed: quality improvement approaches, requirement errors, other software errors, and human errors. The errors described in the answers to Questions 1-3 were collected, analyzed and combined into a Requirement Error Taxonomy with the objective of making the taxonomy simple and easy to use yet comprehensive enough to be effective.

The development of the requirement error taxonomy consisted of two step. First, the detailed list of errors identified during the literature search, and described in the answers to questions 1-3, was grouped into fourteen detailed requirement error classes. Second, the error classes were grouped into three high-level requirement error types. These high-level error types were also found during the literature search. Section 3.4 details the process of developing the Requirement Error Taxonomy and how the taxonomy was constructed.

3.4 Developing the Requirement Error Taxonomy

The taxonomy will guide future error detection research by addressing limitations in the existing quality improvement approaches. The errors identified from the software engineering and psychology fields were collected, analyzed for similarities, and grouped based on similar characteristics (symptoms) into the taxonomy to support the identification of additional related errors when an error is found. An important constraint while grouping the requirement errors was to keep the error classes as orthogonal as possible.

Section 3.4.1 first describes each error class. Then it provides a table of the specific errors from the literature search that are part of that error class. Finally, it gives

an example of an error from that class, along with a fault likely to be caused by that error. The examples are drawn from two sample systems. For the sake of space, only a brief overview of each system is provided here:

- *Loan Arranger System (LA)*: The LA application supports the business of a loan consolidation organization. This type of organization makes money by purchasing loans from banks and then reselling those loans to other investors. The LA allows a loan analyst to select a bundle of loans that have been purchased by the organization that match the criteria provided by an investor. These criteria may include amount of risk, principal involved and expected rate of return. When an investor specifies investment criteria, the system selects the optimal bundle of loans that satisfy the criteria. The LA system automates information management activities, such as updating loan information provided monthly by banks.
- *Automated Ambulance Dispatch System (AAD)*: This system supports the computer-aided dispatch of ambulances to improve the utilization of ambulances and other resources. The system receives emergency calls, evaluates incidents, issues warnings, and recommends ambulance assignments. The system should reduce the response time for emergency incidents by dispatching decisions based on recommendations made by system [4, 48].

3.4.1 *Developing the Requirement Error Classes.*

Tables 3.7 – 3.20 show each error class along with the specific errors (from the software engineering and human cognition fields) that make up that class.

The **Communication Errors** class (Table 3.7) describes errors that result from poor or missing communication among the various stakeholders involved in developing the requirements.

Error: Customer did not communicate that the LA system should be used by between one and four users (loan analysts) simultaneously.

Fault: Omitted functionality because the requirements specify operations as if they are performed by only one user at a time

Table 3.7

COMMUNICATION ERRORS

Inadequate project communications	[20]
Changes in requirements not communicated	[38]
Communication problems, lack of communication among developers and between developers and users	[55]
Communication problems	[51]
Poor communication between users and developers, and between members of the development team	[43]
Lack of communication between sub teams	[11]
Communication between development teams	[13]
Lack of user communication	[13]
Unclear lines of communication and authority	[36]
Poor communication among developers involved in the development process	[8, 22]
Communication problems, information not passed between individuals	[32]
Communication errors within a team or between teams	[52]
Lack of communication of changes made to the requirements	[52]
Lack of communication among groups of people working together	[68]

The **Participation Error** class (Table 3.8) describes errors that result from inadequate or missing participation of important stakeholders involved in developing the requirements.

Error: Bank lender, who was not involved in the requirements process, wanted the LA application system to handle both fixed rate loans and adjustable rate loans.

Fault: Omitted functionality as requirements only consider fixed rate loans.

The **Domain Knowledge Error** class (Table 3.9) describes errors that occur when requirement authors lack knowledge or experience about the problem domain.

Error: Requirement author lacks knowledge about the relative priority of emergency types within the AAD domain.

Fault: The functionality is incorrect because the requirements contain the wrong ambulance dispatch algorithm.

Table 3.8

PARTICIPATION ERRORS

No involvement of all the stakeholders	[42]
Lack of involvement of users at all times during requirement development	[43]
Involving only selected users to define requirements due to the internal factors like rivalry among developers or lack of the motivation	[36, 37, 85]
Lack of mechanism to involve all the users and developers together to resolve the conflicting requirements needs	[20]

Table 3.9

DOMAIN KNOWLEDGE ERRORS

Lack of domain knowledge or lack of system knowledge	[8, 12, 37, 38, 51, 97]
Complexity of the problem domain	[8, 13, 18, 39]
Lack of appropriate knowledge about the application	[22]
Complexity of the task leading to misunderstandings	[37]
Lack of adequate training or experience of the requirement engineer	[38]
Lack of knowledge, skills, or experience to perform a task	[68]
Some properties of the problem space are not fully investigated	[30]
Mistaken assumptions about the problem space	[52]

The **Specific Application Error** class (Table 3.10) describes errors that occur when the requirement authors lack knowledge about specific aspects of the application being developed (as opposed to the general domain knowledge).

Error: Requirement author does not understand the order in which status changes should be made to the ambulances in the AAD system.

Fault: The requirements specify an incorrect order of events, the status of the ambulance is updated after it is dispatched rather than before, leaving a small window of time in which the same ambulance could be dispatched two times.

Table 3.10

SPECIFIC APPLICATION ERRORS

Lack of understanding of the particular aspects of the problem domain	[55, 60]
Misunderstandings of hardware and software interface specification	[52]
Misunderstanding of the software interfaces with the rest of the system	[52]
User needs are not well understood or interpreted while resolving conflicting requirements	[56]
Mistakes in expression of the end state or output expected	[59]
Misunderstandings about the timing constraints, data dependency constraints, and event constraints	[52-53]
Misunderstandings among input, output and process mappings	[72]

The **Process Execution Error** class (Table 3.11) describes errors that occur when requirement authors make mistakes while executing the requirement elicitation and development processes regardless of the adequacy of the chosen process.

Error: Misunderstanding of the ordering of the transactions involving the creation or deletion of records in the database for the LA system.

Fault: The requirements incorrectly specify how the LA should handle transactions. They state that transactions should be resolved based on the order in which the processing completes rather than in the order in which the requests were received.

Table 3.11

PROCESS EXECUTION ERRORS

Mistakes in executing the action sequence or the requirement engineering process, regardless of its adequacy;	[19, 33, 59, 77]
Execution or storage errors, out of order sequence of steps and slips/lapses on the part of people executing the process	[33, 37, 61]

The **Other Human Cognition Error** class (Table 3.12) describes other errors that result from constraints on the cognitive abilities of the requirement authors.

Table 3.12

OTHER HUMAN COGNITION ERRORS

Mistakes caused by adverse mental states, loss of situation awareness	[32, 66, 85]
Mistakes caused by ergonomics or environmental conditions	[8]
Constraints on humans as information processors e.g., task saturation	[37]

The **Inadequate Method of Achieving Goals and Objective Error** class (Table 3.13) describes errors that result from selecting inadequate or incorrect methods for achieving the stated goals and objectives.

- Error:* The requirements engineering misunderstood that some crucial functionality had to be delivered before other functionality, so s/he chose the waterfall lifecycle rather than an incremental one.
- Fault:* Required functionality cannot be delivered on time to the customer.

Table 3.13

INADEQUATE METHOD OF ACHIEVING GOALS AND OBJECTIVES ERRORS

Incomplete knowledge leading to poor plan on achieving goals	[37]
Mistakes in setting goals	[37]
Error in choosing the wrong method or wrong action to achieve goals	[46-47]
Some system-specific information was misunderstood leading to the selection of wrong method	[32]
Selection of a method that was successful on other projects	[19, 65]
Inadequate setting of goals and objectives	[71]
Error in selecting a choice of a solution	[52]
Using an analogy to derive a sequence of actions from other similar situations resulting in the wrong choice of a sequence of actions	[30, 37, 59, 67]
Transcription error, the developer understood everything but simply made a mistake	[55]

The **Management Error** class (Table 3.14) describes errors that result from inadequate or poor management processes.

Error: In the LA system, the same requirement engineer is assigned to document the *borrower's risk requirement* and the *loan's risk requirements*. These contrasting tasks result in a mental lapse when understanding the inputs required to successfully produce the risk estimates.

Fault: The functionality for calculating risk is incorrect.

Table 3.14

MANAGEMENT ERRORS

Poor management of people and resources	[32, 68]
Lack of management leadership and necessary motivation	[24]
Problems in assignment of resources to different tasks	[13]

The **Requirement Elicitation Error** class (Table 3.15) describes errors that result from the use of an inadequate requirement elicitation process.

Error: In the AAD system, the requirements engineers are not able to elicit requirements about system response time for emergency incidents or about error handling.

Fault: Performance and other non-functional requirements are omitted.

Table 3.15

REQUIREMENT ELICITATION ERRORS

Inadequate requirement gathering process	[11]
Only relying on selected users to accurately define all the requirements	[36]
Lack of awareness of all the sources of requirements	[22]
Lack of proper methods for collecting requirements	[13]

The **Requirement Analysis Error** class (Table 3.15) describes errors committed during the requirement analysis process.

Error: In the LA system, the analysis process was not able to identify how the system should respond if multiple transactions are requested that include the same loan before the system is updated.

Fault: The requirements omit this situation leaving it undefined and possibly erroneous.

Table 3.16

REQUIREMENT ANALYSIS ERRORS

Incorrect model(s) while trying to construct and analyze solution	[46-47]
Mistakes in developing models for analyzing requirements	[7]
Problem while analyzing the individual pieces of the solution space	[13]
Misunderstanding of the feasibility and risks associated with requirements	[85]
Misuse or misunderstanding of problem solution processes	[7]
Unresolved issues and unanticipated dependencies in solution space	[12]
Inability to consider all cases to document exact behavior of the system	[55]
Mistakes while analyzing requirement use cases or scenarios	[7, 73]

The **Requirement Traceability Error** class (Table 3.17) describes that result from an inadequate or incomplete requirement traceability process.

Error: In the LA system, a requirement describing the ability of loan analyst to change the borrower information can not be traced to any user need.

Fault: An extraneous requirement is included that could result in extra, unnecessary work for the developers.

Table 3.17

REQUIREMENT TRACEABILITY ERRORS

Inadequate/poor requirement traceability	[[13, 39, 71]
Inadequate change management, including impact analysis of changing requirements	[24]

The **Requirement Organization Error** class (Table 3.18) describes errors committed while organizing the requirement during the documentation process.

Error: When creating the requirements document for the LA system, the requirement engineer does not use any type of logical organization of the requirements.

Fault: Because the requirements are not grouped logically, a requirement about how to display the results of a report was omitted from the requirements document.

Table 3.18

REQUIREMENT ORGANIZATION ERRORS

Poor organization of requirements	[38]
Lapses in organizing requirements	[19]
Ineffective method for organizing together the requirements documented by different developers	[7]

The **No Use of Documentation Standard Error** class (Table 3.19) describes errors committed because the requirement author did not use a standard for documenting the requirements.

Error: In documenting the LA application system, the IEEE standard was not used.

Fault: Requirements about system scope and performance were omitted.

Table 3.19

NO USE OF STANDARD FOR DOCUMENTING ERRORS

No use of standard format for documenting requirements	[38]
Different technical standard or notations used by sub teams for documenting requirements	[36]

The **Specification Error** class (Table 3.20) describes general errors that can occur while specifying the requirements regardless of whether the developers correctly understood the requirements.

Error: In the LA application system, the requirement author understood the difference between regular loans (i.e., for amount \leq \$275,000) and jumbo loans (i.e., for amount $>$ \$275,000), but while documenting the requirements, s/he recorded the same information for both type of loans.

Fault: The requirements for the jumbo loans incorrectly specify exactly the same behavior as for regular loans.

Table 3.20

SPECIFICATION ERRORS

Missing checks (item exists but forgotten)	[11]
Carelessness while organizing or documenting requirement regardless of the effectiveness of the method used	[6, 7]
Human nature (mistakes or omissions) while documenting requirements	[81, 97]
Omission of necessary verification checks or repetition of verification checks during the specification	[46, 47]

3.4.2 *Developing Requirement Error Types*

In order to make the taxonomy understandable and usable for developers, it includes 2 levels. The taxonomy organizes the error classes, from Section 3.4.1, into three high-level types: People Errors, Process Errors, and Documentation Errors. These high-level types can be found in the literature. The remainder of this section describes the three error types and the distribution of the error classes over three error types. The different high-level error types found in the literature are shown in Table 3.21, along with their source. The error types on this information are described in Table 3.21.

Table 3.21

REQUIREMENT ERROR TYPES

Reference	Error Types
Card et al., [20]	1. Incomplete or Wrong <i>methods</i> , 2. Clumsy or defective <i>tools and environment</i> , 3. <i>People</i> who lack adequate training or understanding, and 4 Incomplete or defective <i>input and requirement</i> .
Jacobs et al., [42]	1. Inadequate <i>Communication</i> ; 2. <i>Process</i> risks, 3. Organizational risks concerning decision authorities, responsibilities; 4. <i>Technology</i> risks concerning methods and tools. Does not mention <i>People risks</i> as a separate type, but are embedded in other risk types.
Mays et al., [55]	1. <i>Oversight</i> of the developer; 2. <i>Education</i> of the developer; 3. <i>Communication</i> failure; and 4. <i>Transcription</i> error (developer knew what to do and understood but simply made a mistake)
Basili et al. [6, 7]	1. Application errors (misunderstanding of the problem domain); 2. Problem-solution errors (not knowing, misunderstanding, or misuse of problem solution processes); 3. Environment errors (misunderstanding of the hardware or software environment); 4. Information Management errors (mishandling of certain procedures); 5. Clerical errors (carelessness).
Beecham et al [13]	1. Organizational Issues (i.e., developer communication, culture, resources, skills, training, user communication); and 2. Technical issues (i.e., complexity of the application, poor user understanding, requirements traceability, undefined requirement process).
Sutcliffe et al [79-81]	1. <i>Communication</i> problems; 2. <i>Social and Organization</i> problems (i.e., user participation, lack of understanding or trained personnel); 3. <i>Politics</i> ; and 4. <i>Technical</i> Problems.
Coughlan et al., [22]	1. Poor communication; 2. Lack of appropriate knowledge or shared understanding; 3. Inappropriate, incomplete, or inaccurate documentation; 4. Lack of systematic process; and 5. Poor management of people or resources.
Lutz et al., [52-53]	1. Program Faults (documented software errors); 2. Human errors (i.e., communication errors, misunderstandings specifications or problem domain); and 3. Process flaws (i.e., flaws in controlling system complexity or development methods).

A major focus of the research in Table 3.21 is the contribution of *people* to the errors. Some researchers have explicitly labeled them as people errors or human errors (i.e., Card et al. [20], and Lutz et al., [52-53]), while others mentioned the error types that are more detailed variants of the people errors (e.g., inadequate communication, errors due to the education of a developer, misunderstandings). These errors are already described in the requirement error classes shown in Section 3.4.1.

Another focus in Table 3.21 concerns the errors related to *process* flaws. Jacobs, et al. [42], and Coughlan, et al. [22], have mentioned errors due to flawed development process, flawed methods and tools. While other researchers have recognized this source of errors, they often combine process errors with other error types. An important aspect of the requirement taxonomy is that it distinguishes between flaws in the process (i.e., process errors) and human contribution to the error (i.e., people errors).

Mays, et al. [55], also listed transcription errors i.e., the developer knew what to do and understood but simply made a mistake. Based on this idea, the taxonomy includes the *documentation* error type to describe errors caused by mistakes in organizing and specifying the requirements regardless of whether the developer understood the requirement.

Therefore, the error types used in the requirement error taxonomy are: *People Errors*, *Process Errors*, and *Documentation Errors*. *People Errors* include errors that originate with fallibilities of the individuals involved in project development. *Process Errors* are caused by mistakes in selecting the means of achieving goals and objectives and focus mostly on the inadequacy of the requirement engineering process.

Documentation Errors are caused by mistakes in organizing and specifying the requirements irrespective of whether the developer understood the requirements. After identifying the 14 error types and their meaning, the next step was to classify them using these three error types. Figure 3.2 shows results of that classification as the full Requirement Error Taxonomy.

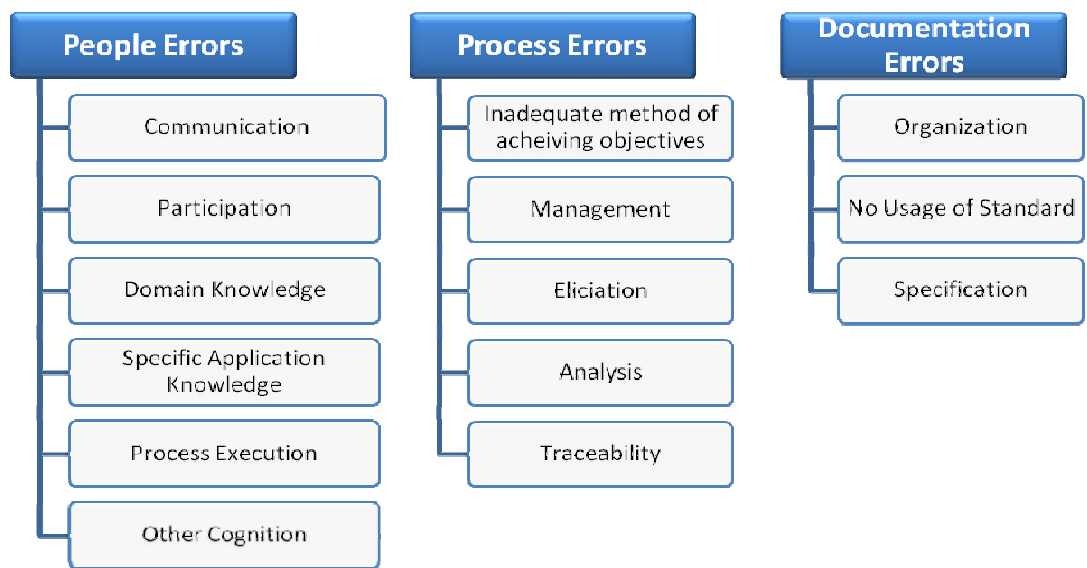


Figure 3.2

Requirement Error Taxonomy

The *Communication* and *Participation* error classes are included in the **People Error** type because they deal with problems among people involved in the development. The *Domain Knowledge* and *Specific Application* error classes are included in the People Error type because they relate to the lack of adequate training provided to people or a person’s misunderstanding of the problem domain. *Process Execution* errors are included

in the People Error type because they relate to errors committed by people during the execution of processes, rather than errors associated with the adequacy of the process itself. Finally, *Other Human Cognition* errors are included in the People Error type because they describe errors that result from constraints on the cognitive abilities of people.

The **Process Error** type includes the error classes that described the errors resulting from flawed methods and tools used for achieving goals and flawed requirement engineering processes. The error classes included in process error were: *Inadequate method for achieving goals and objectives*, *Requirement management* errors, *Requirement Elicitation* errors, *Requirement Analysis* errors, and *Requirement Traceability* errors.

Finally, the **Documentation Error** type includes errors that result from the process of documenting the information about the requirements. They do not concern misunderstandings of the actual requirements, those are people errors. The three error types that are related to this problem are *Specification Errors*, *No Usage of Standard Errors*, and *Organization Errors*.

3.5 *Empirically Validating the Requirement Error Taxonomy*

Once the requirement error taxonomy was developed, the next step was to validate its usefulness and completeness. This evaluation was performed in a controlled classroom environment at Mississippi State University (MSU). The first validation goal was to ensure that the error classes were clearly described, useful, and complete. The

second validation goal was to ensure that developers could use the information provided by the requirement error taxonomy to increase their defect detection effectiveness during the inspection process.

The validation process consisted of planning and conducting three experiments as shown in Figure 3.3: 1) a repeated-measures experiment, 2) a non-equivalent control group quasi-experiment, and 3) an observational group experiment.

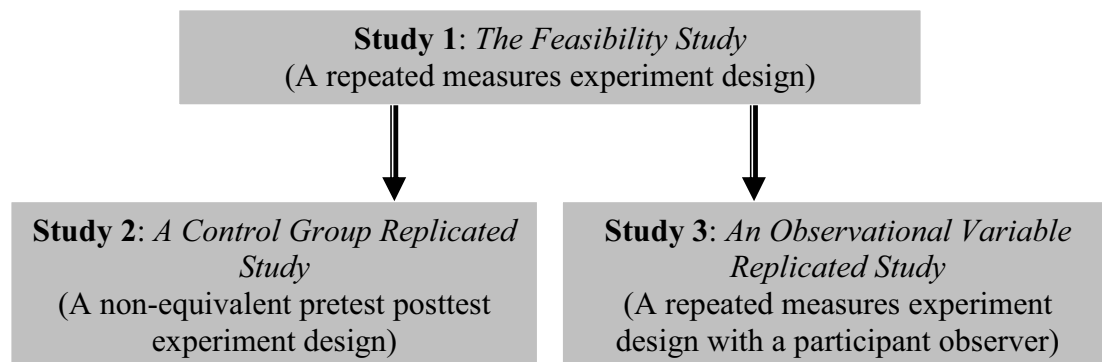


Figure 3.3

The Sequence and Design of Three Controlled Empirical Studies to Evaluate the Requirement Error Taxonomy

In the repeated measures experiment, subjects were asked to first inspect a requirements document (which they created) to find as many faults as possible. These subjects were then trained in the use of error abstraction and classification. After the training, the subjects abstracted errors from the faults they detected. Then, with an understanding of the errors they committed, they re-inspected the requirements to find any additional faults, related to those errors, which were missed during the first

inspection. The additional faults found during the re-inspection were then evaluated to determine the impact of the error taxonomy [86, 87]. This study was originally planned and conducted as a part of my Master's Thesis work. However, the data collected in this study was reanalyzed during this dissertation.

The second study, the non-equivalent control group quasi-experiment, was used to understand how much of the increase in effectiveness seen during the re-inspection process is due to the requirement error taxonomy and how much is due simply to performing a re-inspection. In this study, the control group performed two inspections using the same approach each time. Any additional faults found by the control group during the second inspection constituted the increase due to re-inspection. The treatment group performed the first inspection in the same way as the control group. Then they learned how to use the error abstraction and classification process. They abstracted errors and used this information for the re-inspection. I compared the results of the treatment group to the results of the control group to determine how much of the increase during the second inspection can be attributed to the use of the error abstraction and classification process. The study is a non-equivalent control group quasi-experimental design because logistics demand that the subjects who made up the control group and subjects who made up the treatment group be drawn from different courses [89].

The third study, an observational variable study, was used to gain insights into the thought process of subjects while using the error abstraction and classification process. This goal was achieved by having a participant observer observe the interactions among the subjects and client during the requirement development meeting and also among the

subjects during inspection meetings. The observational data was collected to provide useful insights that could not be obtained from the data collected in the first two studies.

Finally, in all of the three studies, in addition to the quantitative data described above, I also collected qualitative data in the form of post-study surveys and interviews. These surveys and interviews provided the developers an opportunity to give feedback on the error classification taxonomy and re-inspection procedure. The developers were asked to make suggestions on how to improve the taxonomy by adding, removing or combining error classes. Each of the three studies and the details of the results from these studies are described in Chapter IV.

The initial requirement error taxonomy developed during my Master's Thesis and the requirement error taxonomy developed using a systematic review approach in this dissertation contained the same error types and classes. The systematic review expanded the error description in some classes.

3.6 Empirically Validating the Use of Capture-Recapture Method in Software Inspections

I have performed several empirical studies related to the use of the Capture-Recapture method in software inspections as shown in Figure 3.4.

In the first study, I analyzed the effect of the *number of inspectors* on the defect estimates produced by Capture-Recapture models. Previous research showed that the Capture-Recapture estimates are affected by the *number of inspectors* and the *number of defects*. However, the impact of these two factors has not been empirically investigated.

Information about the minimum number of inspectors required to achieve satisfactory estimates is particularly relevant to software organizations to answer the question: *How many reviewers should be included in the inspection?* I analyzed the effect that the *number of inspectors* had on the capture-recapture estimates using data drawn from an inspection performed by 73 Microsoft professionals on a requirement document that was seeded with thirty realistic defects. A major contribution of this study is that it provided a detailed analysis about the number of inspectors that are needed to obtain satisfactory defect estimates, taking into account both accuracy and precision. The project managers can use this result to plan and manage the software inspections in their own organizations [90].

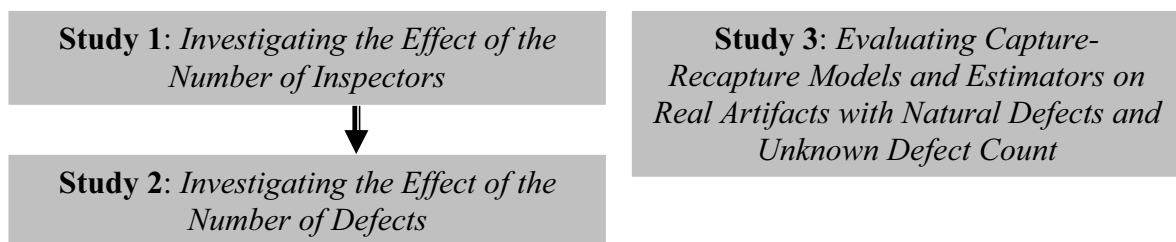


Figure 3.4

Framework of the Studies Conducted to Evaluate the Capture-Recapture Models and Estimators

Following-up from the first study that investigated the effect of the *number of inspectors*, I performed a second study to evaluate the effect the *number of defects* has on the estimates produced by capture-recapture models. The results from this study provided results about the minimum number of defects that have to be present in an artifact before

the capture-recapture estimators can be used [92]. I performed a more detailed analysis of the general trends in the performance of the capture-recapture estimates by evaluating the re-inspection decisions for a different number of defects found during an inspection. This study provides additional insights into how project managers can use the Capture-Recapture estimators to manage real projects in their organizations. I also combined the major analysis and results from the above capture-recapture studies with an additional analysis (to understand the combined effect of the *number of inspectors* and the *number of defects*) to build a body of knowledge that will guide the project managers and software developers on how to cost-effectively use the capture-recapture method in their organizations [93].

While biologists have comprehensively evaluated the capture-recapture models in real settings, most software engineering researchers have used artifacts with a known number of seeded defects. Before using the Capture-Recapture method on live projects to support the re-inspection decisions, an evaluation of the Capture-Recapture models with an unknown number of naturally occurring defects is imperative. I conducted a third study to comprehensively evaluate the Capture-Recapture models on real software artifacts that contained natural defects made during their development. The artifacts were inspected twice (without any modifications or correction between inspections) which allowed the analysis of the Capture-Recapture estimator's ability to make correct re-inspection decisions after each inspection cycle. This study used all the available Capture-Recapture estimators, some of which were used for the first time in the software engineering context. A major result from this study showed that the Capture-Recapture

estimators can be used along with the subjective opinion of the inspectors to correctly decide on the need for re-inspection under realistic conditions. Also, this study recommended the best Capture-Recapture model and Capture-Recapture estimator to be used [91].

The details of the design of the studies and the results from these studies along with the major conclusions are discussed in Chapter V.

CHAPTER IV

EMPIRICAL VALIDATION OF THE REQUIREMENT ERROR TAXONOMY

To evaluate the usefulness of the requirement error taxonomy and the error abstraction process, I conducted three controlled experiments at Mississippi State University (MSU). While each study focused on the same basic hypotheses, the designs of later studies were slightly modified based on the lessons learned during the earlier studies. The overview of these studies is described in Section 3.5. This chapter first discusses the hypotheses and variables common to all studies. Then it provides details of the studies along with the major results.

Rather than discussing these studies in the order in which they occurred (see Figure 3.3), I have grouped them based on study type. Study 2 is a classic control-group study and is therefore presented alone in Section 4.2. Study 1 and Study 3 evaluated the use of the requirement error taxonomy in the context of a real world project developed by university students. Because of their similarities, they are presented together in Section 4.3 to reduce repetition of information.

4.1 Study Hypotheses and Variables

Table 4.1 contains the five hypotheses explored in each of the three studies. Hypothesis 5 investigates the effect of a series of independent variables on the dependent variables. Table 4.2 provides a detailed definition of each of these variables.

Table 4.1

STUDY HYPOTHESES

Hypothesis #1	The <i>error abstraction and classification</i> approach improves the effectiveness (number of faults) and efficiency (faults per hour) of inspection teams and individual inspectors.
Hypothesis #2	The <i>requirement error taxonomy</i> is useful for helping inspectors find errors and faults.
Hypothesis #3	The <i>requirement error taxonomy</i> provides important insights into the major source(s) of requirement faults.
Hypothesis #4	The contributions from <i>cognitive psychology</i> help inspectors locate more faults.
Hypothesis #5	Other <i>independent variables</i> (process conformance, pre-test, training usefulness, effort spent and difficulty level) affect the individual performance (effectiveness and efficiency) during the error abstraction and classification process.

Table 4.2

INDEPENDENT AND DEPENDENT VARIABLES

INDEPENDENT VARIABLE	Definition
Process conformance	Measures how closely subjects follow the error abstraction, classification and re-inspection steps
Pre-test	Measures the performance of subjects on using the requirement error taxonomy during an in-class exercise
Training usefulness	Measures the perceived usefulness of training procedure for each subject
Effort spent	Time spent during error abstraction and classification process (i.e., error abstraction, error classification, and re-inspection)
Difficulty level	Measures the degree of difficulty the students perceived while performing the experimental tasks
Dependent Variable	Definition
Effectiveness	The number of faults found
Efficiency	The number of faults found per hour

4.2 Study 2: A Control Group Replicated Study

This study utilized a non-equivalent control group pretest-posttest quasi-experiment design to understand whether faults found during the re-inspection step are due to use of the error abstraction process and the requirement error taxonomy or due simply to re-inspecting an artifact. This study evaluated the requirement error taxonomy that was derived from the systematic review process described in Chapter III.

4.2.1 Study Design

There were eighteen subjects who were graduate students enrolled in one of two courses, Software Verification and Validation (V&V) or Empirical Software Engineering (ESE) at MSU. The V&V course covered quality improvement approaches including

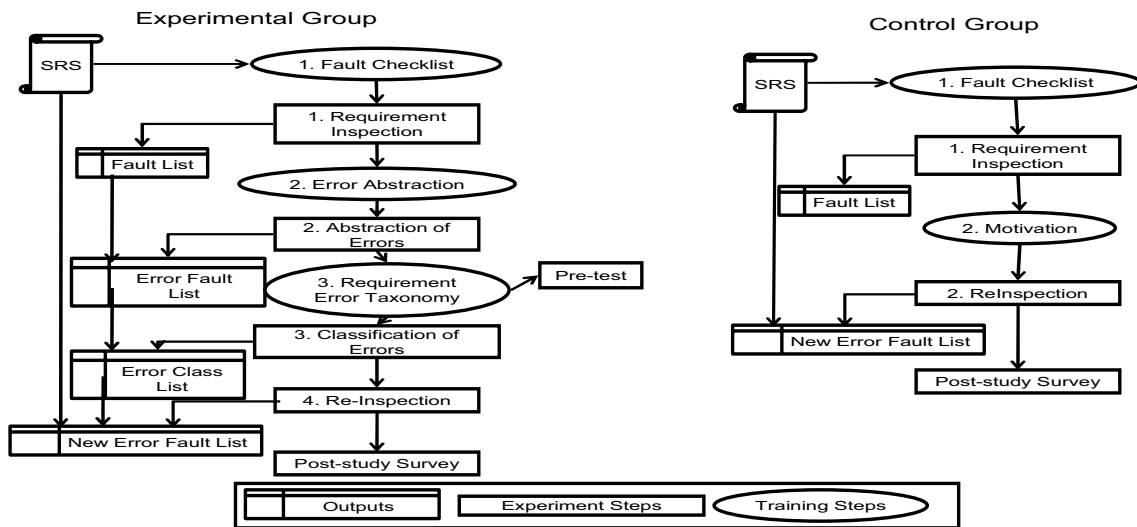


Figure 4.1

Methodology for Study 2

software inspections. The goal of the ESE course was to teach empirical study design and data analysis. During this two-week study, the subjects inspected a requirements document for a data warehouse system developed by the Naval Oceanographic Office. The subjects did not develop the document, nor did they have access to the developers.

The subjects were divided into a control group, which included 9 students from

Table 4.3

STEPS PERFORMED BY EXPERIMENT GROUP AND CONTROL GROUP

	Control Group	Experiment Group
<i>Training 1 – Fault Checklist</i>	Trained on how to use checklist to find faults	
<i>Step 1 - Requirements Inspection</i>	Each subject inspected the requirements to identify faults.	
<i>Training 2 – Motivation / Error Abstraction</i>	Motivation – subjects were informed that faults remained in the document.	Error Abstraction- subjects were trained on how to abstract errors from faults.
<i>Step 2 – Abstraction of Errors</i>	N/A	Each subject abstracted errors from their own fault list from Step 1.
<i>Training 3 – Requirement Error Taxonomy</i>	N/A	The taxonomy was explained. Subjects were taught how to classify errors. They were given an in-class error classification exercise. The exercise was debriefed to ensure the students understood the classification process. Finally, the subjects were taught how to use the classified errors to guide re-inspection.
<i>Step 3 – Classify Errors</i>	N/A	Subjects classified the errors they identified during Step 2.
<i>Step 4 – Re-inspection</i>	Using the fault checklist from Step 1, each subject re-inspected the requirements.	Each subject used their own classified errors from Step 3 to re-inspect the requirements.
<i>Post-study Survey</i>	Focused on the use of the checklist and the quality of the requirements	Focused on gathering feedback about the error abstraction and classification process
<i>In-Class Discussion</i>	In-class discussion held with all subjects to gather additional feedback	

the V&V course and an experiment group, which included 8 students from the ESE course. Four of the 17 subjects were enrolled in both courses. To balance the groups, these subjects were assigned to only one group and were not aware of activities of other group. The experiment steps and training lectures for each group are shown in Figure 4.1, and detailed in Table 4.3

4.2.2 Data Analysis and Results

The results are organized around the five hypotheses presented in Table 4.1. In all cases, an alpha value of 0.05 was used to judge statistical significance.

4.2.2.1 Hypothesis 1: Improve Effectiveness and Efficiency

In terms of effectiveness, Figure 4.2 shows that during the first inspection, there was no significant difference between the control group (avg. of 18 faults) and the

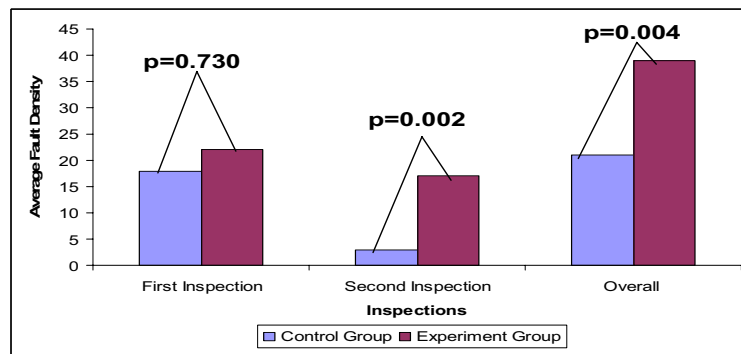


Figure 4.2

Average Number of Faults Found by Experiment and Control Group at 1st, 2nd, and Total Inspection

experimental group (avg. of 22 faults). However, during the second inspection, the experimental group (avg. of 17 new faults) was significantly more effective than the control group (avg. of 3 new faults) [$p = .002$]. As a result, the experimental group was significantly more effective overall (i.e. total number of the defects found in the first inspection plus the second inspection) than the control group ($p = 0.004$).

Efficiency is calculated as: *faults/hour*. For the experimental group, this metric includes the time spent abstracting and classifying errors along with the time spent performing the inspection. For the control group, it includes only the time spent inspecting. There was no significant difference in efficiency between the two groups for either inspection, although the experimental group was more efficient during the second inspection. Because the experimental group was not significantly more efficient, it is possible that the increase in effectiveness was due to extra effort. But, an analysis of covariance showed that the amount of time did not have a significant effect on effectiveness.

Because I conducted four analyses on the same dataset, I introduced the Bonferroni correction to reduce the likelihood of making a Type I error. Therefore, the alpha value was reduced to .013 ($.05/4$). Even using this smaller alpha value, the experimental group is still significantly more effective than the control group.

4.2.2.2 *Hypothesis 2: Usefulness of Requirement Error Taxonomy*

The subjects used a 5-point Likert scale (0- Very Low, 1- Low, 2- Medium, 3- High, or 4- Very High) to evaluate the requirement error taxonomy on ten attributes:

simplicity, usability, orthogonality, usefulness, understandability, intuitiveness, comprehensiveness, uniformity across products, adequacy of error classes, and ease of classifying errors. For each attribute a one-sample t-test was conducted to determine whether the mean was significantly greater than 2 (the mid point of the scale). Five attributes were significantly positive: usefulness ($p = .033$), understandability ($p = .033$), uniformity across products ($p = .049$), adequacy of error classes ($p = .033$), and ease of classifying errors ($p = .033$). The other attributes were also rated positive but not significantly. Overall, the subjects viewed the requirement error taxonomy favorably.

Table 4.4

INSIGHTS PROVIDED BY THE REQUIREMENT ERROR TAXONOMY

Variable	PEOPLE ERROR	PROCESS ERROR	DOCUMENTATION ERROR	P-value
Total errors	41%	25%	34%	0.052
Total Faults	35%	19%	46%	0.002
Redundant Faults	52%	18%	30%	0.004
Time-consuming faults	60%	13%	27%	0.004
Multiple faults	44%	30%	26%	0.002
Important faults	62%	22%	16%	<0.001
Severe faults	53%	28%	19%	<0.001

4.2.2.3 Hypothesis 3: Insights into the Major Source(s) of Requirement Faults

This hypothesis investigates three major questions:

- *Do the three error types make significantly different contributions to the overall defect rate?*

The first two rows in Table 4.4 show the distribution of errors and faults. *People errors* were most common, while *documentation errors* actually led to more faults. The chi-square test confirms that the error types made significantly different contributions to fault injection.

- *Is each of the 14 error classes valid and necessary?*

The results showed that there was at least one fault was caused by an error from each error class. Therefore, each error class is valid and necessary.

- *Which error type(s) are the major source of redundant faults, time consuming faults, multiple faults, important faults, and severe faults?*

Redundant faults are faults that were found by more than one subject. *Time consuming faults* are faults that took longer than the average time (12 minutes) to locate. *Errors that cause multiple faults* are errors that are the source of more than one fault. The *importance* attribute has five levels (0-not important to 4-highly important) and the *severity* attribute has four levels (0-not severe to 3-will cause failure). The results presented in Table 4.4 included levels 2, 3, and 4 levels for the *importance* attribute (ranging from important to highly important) and levels 2 and 3 for the severity attribute.

The bottom 5 rows in Table 4.4 show the results. Again a chi-square test is used to evaluate whether the distribution is significantly different from uniform. For each variable, *people errors* caused significantly more faults than *process* or *documentation* errors. Even though documentation errors led to more faults, those faults tended to be less important and severe, according to the subjects' rankings

4.2.2.4 Hypothesis 4: Usefulness of Cognitive Psychology Research

A portion of the error types in the error taxonomy were derived from cognitive psychology research. The data from this study indicated that 25% of the errors found could be classified into one of these error types. These errors in turn led to 21% of the faults.

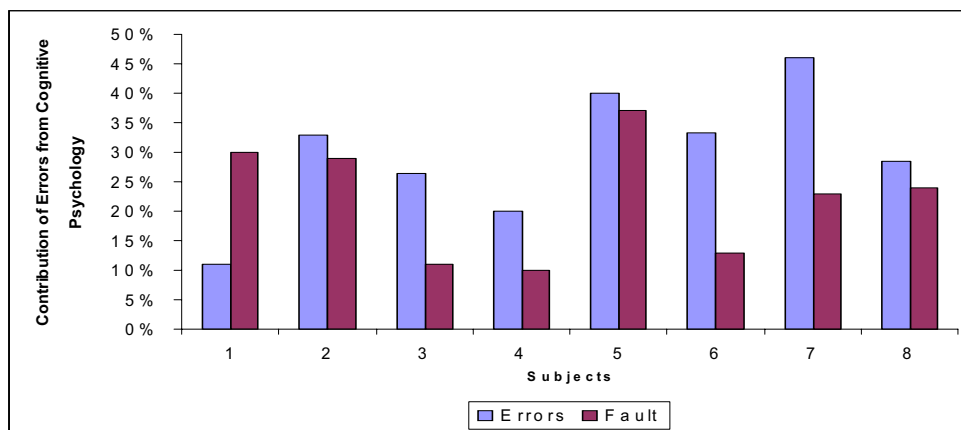


Figure.4.3

Contribution of Cognitive Psychology to Overall Error and Faults found by Eight Subjects in the Experiment Group

Figure 4.3 shows that these errors were not isolated to a small number of subjects. Rather they were spread across all subjects. This result indicates that cognitive psychology research made a useful contribution to the requirement error taxonomy.

4.2.2.5 Hypothesis 5: Effect of Independent Variables

The effects of the five independent variables in Table 4.2 on subject effectiveness were analyzed using a linear regression analysis. The results, shown in Table 4.5 indicate

that *Process Conformance*, *Pre-test* and *Effort Expended* all showed a significant positive correlation to effectiveness during re-inspection. This result means that subjects who followed the process, understood the taxonomy and invested an appropriate amount of effort were more effective. Neither *Training Usefulness* nor *Difficulty Level* was significantly correlated with effectiveness.

Table 4.5

EFFECT OF INDEPENDENT VARIABLES ON SUBJECT’S EFFECTIVENESS

Variable	Description	r ²	p-value
<i>Process Conformance</i>	Subjects rated themselves from 1-5 for Error Abstraction, Error Classification, and Re-inspection (Steps 2-4). The median value was significantly correlated to effectiveness during re-inspection	.528	.041
<i>Pre-test</i>	During Training 3, the subjects classified a set of example errors. The number of correctly classified errors was significantly correlated to effectiveness during re-inspection.	.601	.024
<i>Effort Expended</i>	Effort was the sum of the effort expended during error abstraction, error classification and re-inspection. Effort showed a significant positive correlation to effectiveness during re-inspection	.484	.044

4.2.3 *Validity Threats*

This section describes the validity threats that were addressed and those that were not addressed in this study.

- *Threats Addressed:* To address conclusion validity, a 5-point Likert scale was used for ratings and nonparametric chi-square tests were used because they minimize the assumptions of the underlying data. To address internal validity, the subjects were not informed of the study goals; therefore they were not biased in the data provided. The subjects were graded based on their participation in the study rather than the actual information they provided. To

increase external validity, the study used a real requirement document developed by professionals.

- *Threats Unaddressed:* In this study, a selection threat does exist because the subjects were allocated to groups based on the course in which they were enrolled in rather than randomly. This threat was reduced by selecting the course (V&V) whose students were more likely to perform better inspections (because of the course material) to be the control group. This choice was made so that if any bias was present, it would be in favor of the control group and not the experiment group. There was an external validity threat because the subjects were graduate students and likely to have different experience and time pressure than professionals in real settings. The last threat that could not be addressed in this study was maturation. The subjects in control group performed two inspections on the same document using the same technique. It is possible that subjects were less motivated during the second inspection. Although, such a result would be a good argument for providing a different approach for the re-inspection to help inspectors stay engaged in the activity.

Table 4.6

SUMMARY OF FINDINGS FROM STUDY 2

Hypothesis	Summary of Findings
<i>H 1</i>	The experimental group (using error taxonomy) was significantly more effective than the control group (using fault checklist) during reinspection (p= 0.002) as well as overall (p= 0.004), with no differences in their efficiencies
<i>H 2</i>	The error taxonomy was significantly favorable relative to usefulness (p=0.033), understandability (p=0.033), uniformity across different products (p=0.049), and containing adequate requirement error classes (p=0.033)
<i>H 3</i>	<i>People errors</i> led to a larger percentage of errors (p=0.052) and <i>documentation errors</i> led to a larger percentage of faults (p=0.002), <i>People errors</i> also led to a significantly larger redundant and, time-consuming faults (p=0.004), multiple faults (p=0.002), and important and severe faults (p=<0.001).
<i>H 4</i>	The experiment group found that 21% of the total faults were caused by cognitive errors; and all subjects found fault(s) caused by them
<i>H 5</i>	Pre-test (p=0.024), overall process conformance (p=0.041), and effort spent (p=0.044) led to the detection of significantly more faults during 2 nd inspection

4.2.4 *Results Summary*

Table 4.6 provides a brief summary of the findings from Study 2 for each of the five hypotheses.

4.3 *Study 1: Feasibility Study and Study 3: An Observational Study*

The goal of Study 1, a repeated measures quasi-experiment, was to understand whether the requirement error taxonomy could be effectively used in a traditional inspection process [87]. It compared the effectiveness of subjects in finding defects recorded in the requirement documents with and without using the taxonomy. Study 1 was originally conducted during my Master's Thesis to evaluate the initial requirement error taxonomy developed using an ad-hoc literature review process [86]. However, the data from Study 1 was reanalyzed in this dissertation.

Study 3 was a replication of Study 1, but with an additional emphasis on qualitative data. It used a *participant observer* to capture firsthand observations about the behavior and interactions among subjects that could not be collected quantitatively. The main goal of Study 3 was to gain insight into the human thought process related to requirement error and faults. Due to similarity in their study designs, Study 1 and Study 3 are described together in following sub-sections.

4.3.1 *Study Designs*

The subjects in both studies were senior-level computer science or software engineering students enrolled in the senior design capstone course during two different

years. During the course, the students interacted with real customers to elicit and document the requirements for the system they later implemented. In each study, the subjects were divided into two independent teams that created their own requirements document.

Table 4.7

STUDY 1 AND STUDY 3: TEAMS, SUBJECTS, AND SYSTEM DESCRIPTION

Study#	SEMESTER	TEAM #	NUMBER OF SUBJECTS	SYSTEM
1	Fall 2005	1-A	8	Starkville theatre system
		1-B	8	Apartment Management
3	Fall 2006	3-A	6	Conference management
		3-B	6	

Table 4.8

DIFFERENCES IN STUDY 1 AND STUDY 3 FROM STUDY 2

Changes from Study 2
Unlike Study 2, the subjects had team meetings after the first inspection, the error classification and the re-inspection to consolidate the individual fault or error lists into a team list
The feedback questionnaire was similar for all 3 studies except that in Study 1, a 3-point scale was used instead of a 5-point scale.
Study 1 was a feasibility study, so it used the initial taxonomy developed as a result of ad-hoc literature review. Studies 2 and 3 used the refined taxonomy derived from the systematic literature review as described in Figure 3 (see Section 3.4.2).
No in-class discussion was held at the conclusion of the study. However, the team leaders were interviewed to gather feedback.

Table 4.7 provides the details of the study. Note that the two teams in Study 3 developed the same system, conference management, but they worked independently.

The studies began after the teams developed the initial requirement specification for their respective systems. These studies contained the same training and steps as used by the experiment group in Study 2, other than the exceptions shown in Table 4.8.

Figure 4.4 illustrates the study design. Because Study 3 is a replication of Study 1, the only difference between the two studies is the addition of a participant observer. The role of the participant observer is shown by the dotted lines and is described in the remainder of this section. During Study 3, I acted as a participant observer during portions of the requirements development and review process. Seamen, et al., point out that observing communication among software developers during team meetings offers

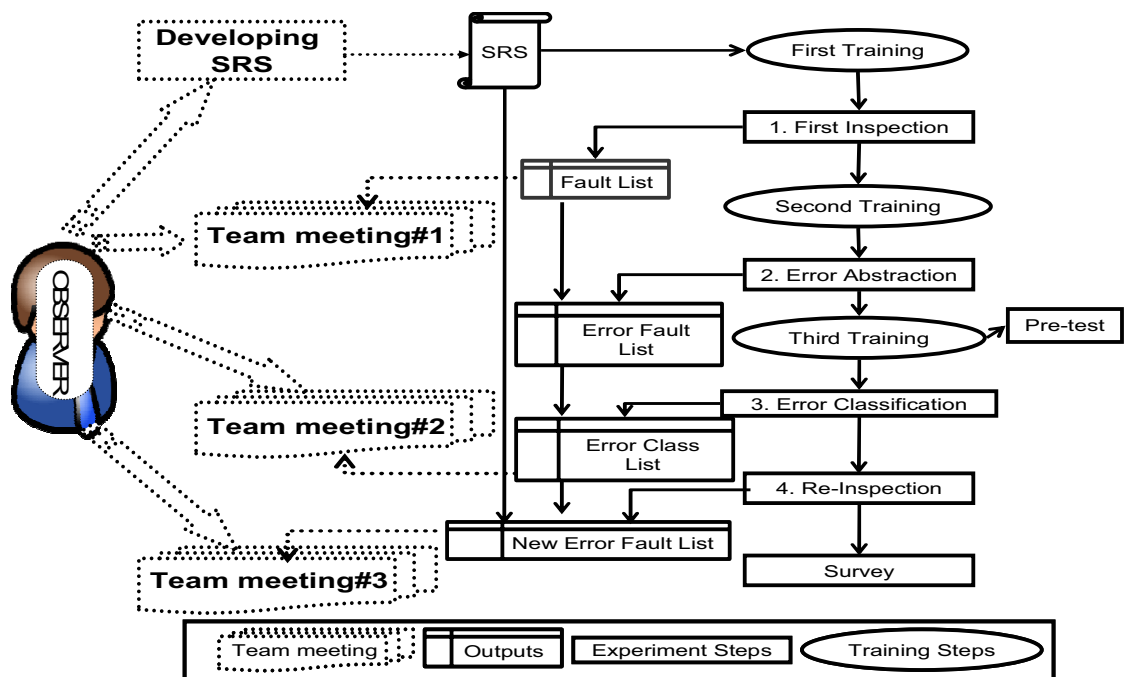


Figure 4.4

Study Operation for Study 1 in Solid Lines and the Additional Observational Aspect in Study 3 in Dotted Lines

an easy yet important approach to understand the thought processes of software developers [69-70]. Therefore, in this study, I gathered *field notes* during: 1) team meetings for requirements gathering, and 2) team meetings for requirement inspection.

Details of the responsibilities of the observer and data collected during the observation are described below:

- *Observations during the requirement gathering team meetings:* During these meetings, the client explained the requirements to all subjects, who were given an opportunity to ask questions for clarification. Later, each team met separately with the client to clarify any remaining issues and gather additional requirements. The goal of observing these meetings was to document any errors that were observed to later check whether the subjects identified these errors during the error abstraction and classification process. The data collected included notes about the communication between the subjects and the client.
- *Observations during the inspection team meetings:* Each team met three times during the inspection process to discuss the faults and errors found individually and to consolidate their individual lists into a team list. These meetings occurred after the first inspection, after the error classification, and after the reinspection. I was present at each meeting for both teams to gather observations about the communication and interactions among subjects and to record additional faults and errors identified during the team meeting.

4.3.2 *Data Analysis and Results*

The results from Studies 1 and 3 are organized around the five hypotheses. Also, the insights gained by the analysis of the observational data in Study 3 are discussed.

4.3.2.1 *Hypothesis 1: Improves Effectiveness and Efficiency*

The effectiveness of each team was analyzed by comparing the number of new faults found during the second inspection (using the error abstraction and classification

process) to the number of faults found during the first inspection (using the fault checklist).

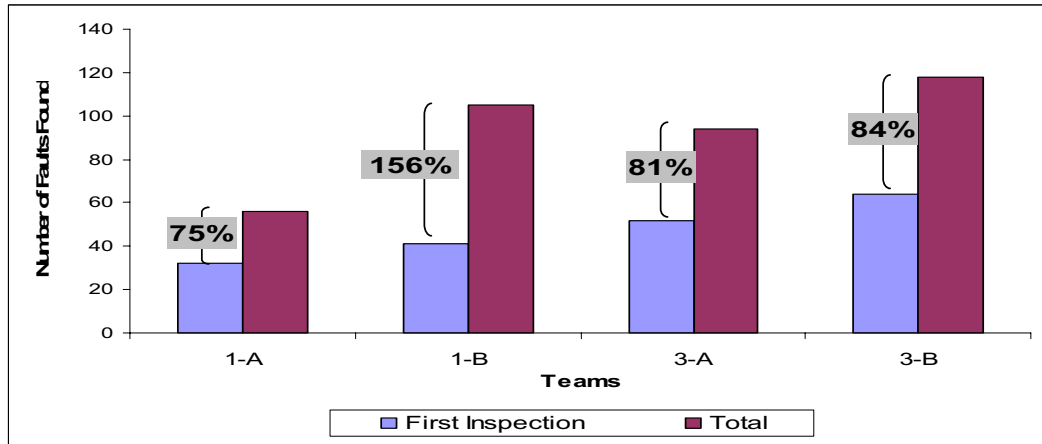


Figure 4.5

Increase in the Number of Faults Found

As Figure 4.5 shows, the teams showed an increase (*faults found during reinspection / faults found during first inspection*) of between 75% and 156% when using the error abstraction and classification process. Therefore, using error abstraction and classification to inform the reinspection was highly effective at helping inspectors detect faults that were missed during first inspection.

This analysis was also performed for each subject to better understand whether this large increase was caused by a few subjects or spread across the entire sample. The result showed that, while some subjects had a larger increase than others, all subjects found new faults during the second inspection. Therefore, all the subjects in teams 1-A, 1-B, 3-A, and 3-B also showed an increase in their effectiveness. We performed a one-sample t-test to determine if the increase was significantly greater than 0 for each team.

The results indicate that it was: $p = .02$ for team 1-A, $p < .01$ for team 1-B, $p = <0.00$ for team 3-A, and team 3-B.

Similar to the results from Study 2 the error abstraction and classification process did not have a significant impact on efficiency. In Study 1, there was a slight decrease in efficiency that was balanced by the increase in effectiveness.

4.3.2.2 *Hypothesis 2: Usefulness of Requirement Error Taxonomy*

The requirement error taxonomy was evaluated using the same ten attributes as used in Study 2. However (as mentioned in Section V.A), the sixteen subjects in Study 1 evaluated the initial version of requirement error taxonomy using a 3-point scale (low-medium-high) whereas, the twelve subjects in Study 3 evaluated the refined version of requirement error taxonomy using a 5-point scale (very low-low-medium-high-very high).

Similar to the analysis performed for Study 2, a one-sample t-test was used to determine whether the ratings were significantly skewed towards “high”. In Study 1 five attributes were significantly positive: usability ($p = .02$), usefulness ($p = .041$) comprehensiveness ($p = .009$), uniformity ($p = .009$) and ease of classifying ($p = .009$). In Study 3, all ten attributes were significantly positive. Two potential reasons that the taxonomy was rated more positively for Study 3 than for Study 1 are: 1) the use of 5-point scale rather than a 3-point scale, and 2) use of refined version of the error taxonomy. These results show that the subjects had a strong positive response to the requirement error taxonomy.

4.3.2.3 Hypothesis 3: Insights into the Major Source(s) of Requirement Faults

The analysis for these studies was conducted in the same way as for Study 2. Table 4.9 shows the distribution of errors and faults among the people, process, and documentation error types. Table 4.9 also shows the p-value from the chi-square test that evaluated whether the distribution was significantly different from uniform.

Table 4.9

CONTRIBUTION OF ERROR TYPES TO OVERALL ERRORS AND FAULTS

Variable	Team	People Errors	Process Errors	Documentation Errors	p-value
Errors	1-A	52%	32%	16%	<0.001
	3-A	50%	30%	30%	0.001
	1-B	59%	24%	17%	<0.000
	3-B	38%	35%	27%	0.379
Faults	1-A	50%	33%	8%	<0.001
	3-A	57%	27%	16%	<0.001
	1-B	78%	9%	13%	<0.000
	3-B	47%	19%	34%	0.003

In these studies, *people errors* led to the most errors and faults, as opposed to Study 2 where *documentation errors* led to more faults

Table 4.10 shows the contribution of the *People*, *Process*, and *Documentation* errors to redundant, time-consuming, multiple, important, severe, important, and severe faults for each team. Note that data regarding *importance* and *severity* was only collected during Study 3. In both studies, *people errors* were significantly more likely to cause

Table 4.10

ADDITIONAL INSIGHTS PROVIDED BY REQUIREMENT ERROR TAXONOMY

Variable	Team	People Errors	Process Errors	Documentation Errors	p-value
Redundant faults	1-A	64%	29%	7%	< 0.001
	1-B	64%	23%	14%	< 0.001
	3-A	63%	21%	16%	< 0.000
	3-B	49%	17%	34%	< 0.000
Time consuming faults	1-A	38%	62%	0%	< 0.016
	1-B	25%	58%	17%	< 0.001
	3-A	24%	68%	8%	< 0.000
	3-B	31%	54%	15%	< 0.000
Multiple faults	1-A	80%	20%	0%	< 0.001
	1-B	60%	36%	4%	< 0.001
	3-A	47%	15%	38%	< 0.000
	3-B	38%	11%	51%	< 0.000
Important faults	3-A	65%	21%	14%	< 0.000
	3-B	52%	39%	9%	< 0.000
Severe faults	3-A	71%	29%	0%	< 0.000
	3-B	58%	37%	5%	< 0.000

redundant faults, multiple faults, important faults and severe faults in all but one case (for team 3-B *documentation errors* were more likely to cause multiple faults).

4.3.2.4 Hypothesis 4: Usefulness of Cognitive Psychology Research

For all four teams, errors related to cognitive psychology accounted for at least 1/5 of the total errors (1-A → 21%, 1-B → 26%, 3-A → 32%, and 3-B → 24%). This result was fairly consistent across the subjects on the teams.

Figure 4.6 show that fourteen of the sixteen subjects in Study 1 found at least one human cognition error. In fact, one subject found only human cognition errors. Figure 9 also shows that only one subject in Study 3 (team 3-B) did not find any human cognition errors. Only two more did not find any faults related to human cognition errors. The results show that, in some cases, errors did not actually lead to detection of additional faults during reinspection while some errors lead to detection of multiple faults. These results support the findings of Study 2 that errors related to cognitive psychology research are important to include in the requirement error taxonomy.

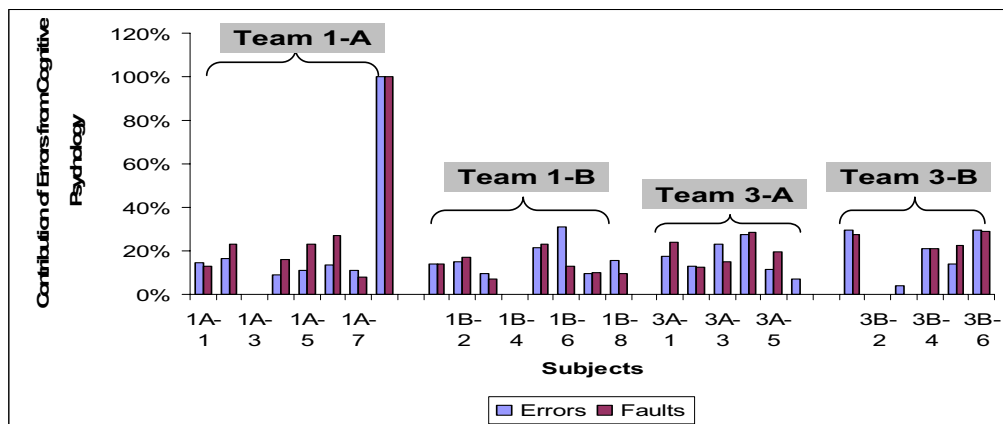


Figure 4.6

Contribution of Cognitive Psychology Research to Overall Error and Faults found by Each Subject in Teams 1-A, 1-B, 3-A, and 3-B

4.3.2.5 Hypothesis 5: Effect of Independent Variables

The same five independent variables analyzed in Study 2 were analyzed in these studies. The goal was to find any significant correlations to effectiveness during the

reinspection. For Study 1, the variables were rated on a 3-point scale, while for Study 3 a 5-point scale was used. A linear regression analysis was used.

The median *process conformance* rating was significantly correlated to effectiveness ($r^2 = .385$, $p = 0.03$ for Study 1 and $r^2 = .439$, $p = 0.019$ for Study 3), meaning that the better a subject followed the process the more effective they were.

Pre-test performance had a weak positive correlation to effectiveness during the reinspection for Study 1 ($r^2 = .366$, $p = .017$). However, for Study 3, the correlation was stronger, but not significant ($r^2 = .548$, $p = .065$).

In Study 1, *effort* had a weak positive correlation to efficiency ($r^2 = .331$, $p = .02$). This result is interesting because it shows that the more effort spent, the more faults/hour the subject found (not just more faults found). There was no correlation found in Study 3.

Usefulness of training was significantly correlated with effectiveness in reinspection for Study 3 ($p = .020$), i.e. subjects who found the training useful were more effective. *Difficulty* did not show any significant correlation.

4.3.2.6 *Additional Insights from Qualitative Data (Study 3)*

During the *requirement gathering team meetings*, I recorded any errors observed during development of the requirement documents. This list of errors did not include those that might have occurred outside of the team meeting. I noted 17 errors made by team 3-A and 22 by team 3-B. When these lists of errors were compared to the final team error list, team 3-A found 13/17 errors and team 3-B found 20/22 errors. All of these errors were identified by subjects during individual inspections prior to the team

meetings. Therefore, the requirement error taxonomy was useful in identifying errors committed during development and useful for helping developers find errors during inspection.

For the data collected during the *inspection team meetings*, four analyses were performed. First, I analyzed whether the meeting led to additional faults and errors being identified by the group. There were two meetings related to faults, meetings 1 and 3, and one meeting related to errors, meeting 2 (after abstraction and classification). During meeting 1, after the first inspection, team 3-A found one new fault and team 3-B zero new faults. During meeting 3, after the reinspection, Team 3-A found five new faults and Team 3-B three new faults. Therefore, meeting 3 seems to have been more effective for both teams. During meeting 2, after error abstraction and classification, Team 3-A found one new error and Team 3-B three new errors. Therefore the team meetings led both teams to identify problems that had been missed by individuals. Specifically, meetings 2 and 3, which were informed by the error abstraction and classification process, were more effective than meeting 1.

Second, the thought processes of the developers as they discussed the inspection results were analyzed. They subsequently found some new faults that were not found during individual reinspection step. For example, one member of Team 3-A indicated that the team leader did not communicate an important requirement properly which lead to incorrect documentation. During this discussion, the team members were able to find the two faults introduced in that requirement. Often during such interactions, other team members contributed to the discussion especially in terms of how such errors affected

other parts of the requirements document. The results of this analysis indicated that after discussing the errors with each other, the subjects better understood the requirement problems and the faults that were likely to result. Therefore, the subjects found additional faults.

Third, the content of the discussions were analyzed. The discussions were more engaging and longer during team meetings 2 and 3, which were informed by the error abstraction and classification process, than during team meeting 1, and are summarized as follows:

- During team meeting 1, the discussion focused mainly on deciding whether the fault(s) found by individuals made sense to the rest of the team. They also spent time eliminating redundant faults and compiling the team fault list. The team members seemed to disagree mostly on the type of fault (i.e., incorrect fact, ambiguous, etc...) with some instances where no clear consensus was reached.
- During team meeting 2, often different team members had classified the same error into different error classes. The discussion focused mainly on agreeing to one compiled list of errors and their classification. While there were initial disagreements, the teams always reached consensus.
- During team meeting 3, the team members discussed the different faults they found which were caused by the same error. While compiling the individual fault lists into a team list, new faults were identified. These faults were discussed and included on the final list. Each teams found more faults in this meeting than in team meeting 1.

Finally, the pattern of discussion was analyzed. Because the team leader's job was to compile the list, he was always the center of the discussion. However, I observed a pattern that after a team member finished describing their own list of errors or faults, they became less interactive. This pattern was more visible for Team 3-B than for Team 3-A. Also, the new faults and errors that arose during the discussions did not originate from a

single person or small group of people. Rather, various team members were involved in identifying different faults or errors.

4.3.3 Validity Threats

This section describes the validity threats that were addressed and those that were not addressed in this study.

- *Threats Addressed:* To address conclusion validity, the threat due to heterogeneity of subjects was controlled because all subjects were drawn from the same course and had same level of education. The conclusion validity threat caused by the use of a 3-point rating scale in Study 1 was addressed by increasing the scale to a 5-point scale in Study 3. To partially address external validity threats, the studies 1 and 3 were done in the context of a capstone course where the subjects were developing a real system for a real client.
- *Threats Unaddressed:* The external validity threat that was unaddressed in Studies 1 and 3 was the fact that the subjects were students rather than professionals.

4.3.4 Summary of Results

Table 4.11 provides a brief summary of the findings from Study 1 and Study 3 for each of the five hypotheses.

4.4 Discussion of Results from Three Studies

This section discusses results of all three studies in light of the five common hypotheses. The goal is to draw conclusions from the series of studies rather than from each individual study. In some cases the general conclusions are the same as those drawn in a single study, but stronger because of common results in all three studies.

Table 4.11

SUMMARY OF FINDINGS FROM STUDY 1 AND STUDY 3

Hypothesis	Findings from Study1	Findings from Study 3
<i>H 1</i>	Improved the effectiveness of teams and individual inspectors, but did not have a significant effect on the efficiency	Significantly improved the effectiveness of teams and individuals, but did not have a significant effect on the efficiency
<i>H 2</i>	The requirement error taxonomy was significantly favorable on 5 out of ten attributes,	The requirement error taxonomy was viewed as significantly favorable on all the ten attributes
<i>H 3</i>	<i>People errors</i> led to a significantly larger percentage of errors, faults, redundant faults, and multiple faults; Faults resulting from <i>process errors</i> took more time to detect as compared to other error types	<i>People errors</i> led to a significantly larger percentage of errors and faults, as well as multiple faults for Team 1; and redundant, important and severe faults for both teams; The <i>observational method</i> revealed that communications during team meetings were productive in discovering more errors and faults; and improved their understanding of requirement errors
<i>H 4</i>	Team 1 and Team 2 found 21% and 26% of total faults using the cognitive errors respectively; and 14 of 16 subjects found one/more of these errors	Team 1 and Team 2 found 32% and 24% of total faults using the cognitive errors; and 11 of 12 subjects found at least one cognitive error, and 9 out of 12 subjects found at least one fault resulting from these cognitive errors
<i>H 5</i>	Pre-test, process conformance, and training on error abstraction affect the effectiveness of subjects. The overall effort spent affects the efficiency	Pre-test, overall process conformance, training on the requirement error taxonomy, and the overall effort spent leads to the detection of significantly more faults during second inspection

4.4.1 Hypothesis 1

The error abstraction and classification approach improves the effectiveness (number of faults) and efficiency (faults per hour) of inspection teams and individual inspectors

For team effectiveness, all three studies showed that using the *error abstraction and classification* process was beneficial. In Studies 1 and 3, each team found a significant number of new faults during the reinspection in which they used the error abstraction and classification process (75% and 156% for teams in Study 1 and 81% and

84% for teams in Study 3). In Study 2, the subjects who used the error abstraction and classification process were significantly more effective during reinspection than those who just inspected the document two times. In terms of individual effectiveness, the results show a consistent benefit for all subjects. Each subject found a significant number of faults during the reinspection.

The results of all three studies showed that the error abstraction and classification process did not significantly affect efficiency. This result is not surprising due to the extra steps required by the error abstraction and classification process.

Combining the results of effectiveness and efficiency, I can conclude that the error abstraction and classification process significantly improved effectiveness while not hurting the efficiency of inspectors compared with the fault checklist method. Because I now know that the error abstraction and classification process is effective, a future step is to investigate ways to improve its efficiency. The students suggested that they might perform better if they were trained in the requirement error taxonomy before performing the error abstraction step. So, a future study will have the subjects directly using the error taxonomy to abstract errors and re-inspect rather than performing the actions in two separate steps as done in these studies

4.4.2 Hypothesis 2

The requirement error taxonomy is useful for inspectors find errors and faults

The data to evaluate this hypothesis came from subjective ratings by the subjects. In Study 1, the subjects evaluated the error taxonomy using a 3-point scale whereas in

Studies 2 and 3, they evaluated the error taxonomy using a 5-point scale.

Table 4.12 shows the ratings for the attributes across all three studies. Attributes that were significantly favorable have a double arrow (\Uparrow), while those that were favorable, but not significant, have a single arrow (\Uparrow). The results show that each attribute was rated significantly positive in at least one of the three studies.

Post-experiment interviews in Studies 1 and 3 and in-classroom discussion in Study 2 provided some additional insights into these results. The subjects believed that the error abstraction and classification process is easier when the inspectors participated in the development of the requirements. This hypothesis will be studied in the future. Second, subjects from all three studies agreed that using the error information helped them better understand the real problems in the requirements document, and that the effort spent during the error abstraction and classification process was worthwhile.

Table 4.12

INSIGHTS PROVIDED BY THE REQUIREMENT ERROR TAXONOMY

Attributes	STUDY 1	STUDY 2	STUDY 3
Simplicity	\Uparrow	\Uparrow	\Uparrow
Usability	\Uparrow	\Uparrow	\Uparrow
Orthogonal	\Uparrow	\Uparrow	\Uparrow
Usefulness	\Uparrow	\Uparrow	\Uparrow
Understandability	\Uparrow	\Uparrow	\Uparrow
Intuitiveness	\Uparrow	\Uparrow	\Uparrow
Comprehensive	\Uparrow	\Uparrow	\Uparrow
Uniformity across products	\Uparrow	\Uparrow	\Uparrow
Adequacy of error classes	\Uparrow	\Uparrow	\Uparrow
Ease of classifying errors	\Uparrow	\Uparrow	\Uparrow

Finally, the subjects agreed that the errors in the requirement error taxonomy were a true representation of the mistakes that can occur during software development. They were in favor of the creation of similar error taxonomies for other lifecycle stages (i.e. architecture/design, coding).

4.4.3 Hypothesis 3

The requirement error taxonomy provides important insights into the major source(s) of requirement faults

One of the insights provided by the requirement error taxonomy was the relative contributions of three error types. In Studies 1 and 3, there were significantly more *people errors* and significantly more faults related to people errors. Conversely, in Study 2, there were more *people errors*, but the *documentation errors* actually lead to significantly more faults. One hypothesis to explain this difference is that participation in the development of the requirements affects the types of errors identified during an inspection. In Studies 1 and 3 where *people errors* led to largest number of faults, the subjects participated in the development of the requirements. Conversely, in Study 2, where *documentation errors* led to largest number of faults, the subjects did not participate in the development of the requirements.

Furthermore, the result showed that a significantly higher number of redundant faults, multiple faults, important faults and severe faults were the results of *people errors*. Further study of the fourteen detailed error classes showed that at least one error from each class was found by some subject(s) in all three studies. This result provides confidence that the error classes in the requirement error taxonomy are valid and provide

a good coverage of the requirement error space. Another insight was that no consistent pattern emerged regarding the types of faults that were caused by different error classes. The results do show that the *people* and *process* errors more often led to faults of ambiguity, and missing functionality. The insights from the participant observation in Study 3 showed:

- The subjects in each team found some, but not all of the errors observed during development of the requirements,
- During the team meetings after the error abstraction and classification process the subjects found more errors and faults than during the meetings after the first inspection, using only the fault checklist,
- The subjects better understood the requirement problems after becoming aware of the errors committed during development of the requirements document, and subsequently found more faults, and
- The subjects followed the error abstraction and classification process closely and communicated well with other subjects during the inspection team meetings.

4.4.4 Hypothesis 4

The contributions from human cognition and psychology help inspectors locate more faults

In all three studies, the subjects found a large percentage of faults that were related to cognitive psychology errors. In Study 1, the two teams found that 21% and 26% of the faults were related to cognitive psychology errors. In Study 2, 21% of the faults were related to cognitive psychology. In Study 3, the two teams found that 32% and 24% of the faults were related to cognitive psychology. Furthermore, almost all of the subjects in each of the three studies found errors and/or faults that were related to cognitive psychology. Overall, these results support the validity of using human

cognition research to better understand fault production and improve the effectiveness of software inspectors. Therefore, the integration of software engineering and cognitive psychology research helps provide a solution to the software quality problem.

4.4.5 Hypothesis 5

Other independent variables (process conformance, performance on a pre-test, usefulness of training, effort, and perceived difficulty) affect the individual performance during the error abstraction and classification process

The correlation between the independent variables and dependent variables differs across the three studies. The major conclusions about the impact of the independent variables on the performance of the inspectors are summarized as follows:

- The ability to correctly classify example errors during a pre-test (an indication that the requirement error taxonomy is understood) is a good predictor of fault detection effectiveness;
- Process conformance will help the subjects to find larger number of faults;
- Effective training on error abstraction and on the use of the error taxonomy for reinspection is necessary for improving the fault detection effectiveness; and
- An increase in the effort spent during the error abstraction and classification process is likely to lead to an increase in fault effectiveness, but can also increase the fault efficiency.

4.4.6 Validity Threats in All Three Studies

The first threat is that all three studies were conducted in university settings. The subjects, undergraduates in Studies 1 and 3 and graduates in Study 2, are likely not representative of professionals. Another threat common to all three studies is that the environment was not representative of a real setting. This threat was minimized in

Studies 1 and 3 by having the students work with real clients to develop requirement documents that were later used to implement systems. Also, in Study 2 students inspected a real software requirement document developed by external software professionals.

4.4.7 Results Summary

Table 4.13 summarizes whether the findings from the three studies support each of the five hypotheses.

Table 4.13

SUMMARY OF FINDINGS FROM ALL THE STUDIES

Study #	Hypothesis 1. (Improves Effectiveness, Efficiency)	Hypothesis 2 (Usefulness of the RET)	Hypothesis 3 (Provides Important Insights)	Hypothesis 4 (Cognitive Psychology Contribution)	Hypothesis 5 (Effect of Independent Variables)
1	For all 3 Studies; Effectiveness- Yes; Efficiency- No	Yes (on 5 attributes)	Yes	Yes	Pre-test, Process Conformance, Training, and Effort Spent affect performance
2		Yes (on 4 attributes)	Yes	Yes	
3		Yes (on all 10 attributes)	Yes	Yes	

CHAPTER V
EMPIRICAL VALIDATION OF CAPTURE RECAPTURE IN SOFTWARE
INSPECTIONS

A series of empirical studies were conducted to evaluate the usefulness of the capture-recapture method for estimating post-inspection defects. While each study had a different design and goal(s), the capture-recapture models and estimators used in all the studies were same as in Section 2.4.1. This chapter provides details of each study along with the major results from each study. The detailed results from each study have been published elsewhere [90-93].

5.1 *Study 1: Investigating the Effect of the Number of Inspectors [90]*

Since the capture-recapture models work based on the amount of overlap in the defects detected by different inspectors, it is unclear what effect a large number of inspectors will have on the performance of the capture-recapture models. Therefore, the primary goal of the study was to understand how the performance of the estimators improves when increasing the number of inspectors. Stated formally, the goal was to:

*Analyze the capture-recapture estimators
For the purpose of characterizing the impact of the number of inspectors
With respect to defect estimation
From the point of view of the project manager and developers.*

5.1.1 Data Set

The data for the capture-recapture analysis was drawn from an earlier inspection study conducted at Microsoft Research. The goal of the original study was to investigate the impact of educational background on the effectiveness of an inspector. Only the information that is relevant to the capture-recapture analysis is provided in this section:

- *Software Artifact*: The artifact inspected during this study was a generic (i.e. non-Microsoft) requirements document describing the requirements for the Loan Arranger system financial system. For use in previous studies, the document was seeded with thirty realistic defects. The defect seeding was done by researchers other than us prior to the design of the capture-recapture study. Therefore defects that were seeded should not provide bias in this study;
- *Software Inspectors*: There were 73 inspectors who were drawn from an internal training course taught by the Microsoft Engineering Excellence group. The goal of course was to teach participants about inspections and their use at Microsoft. The participants were drawn from all major product groups across Microsoft. About 70% had bachelor's degrees with the other 30% having Master's degrees. On average, the participants had about 2 years of experience working in the field;
- *Inspection Process*: First, the participants received training on the basic concepts involved in an inspection process. Then, the participants performed their own inspection of the Loan Arranger requirements document. To guide their review of the document, the participants used a standard fault checklist. During the inspection, each participant worked alone to identify and record as many faults as possible. They were given seventy minutes to complete the inspection task. At the conclusion of the inspection, the 73 individual fault lists were collected and processed. The processing involved determining which of the thirty seeded faults were found by each participant. It is this information that was used as raw data for the capture-recapture study described in the remainder of this section.

5.1.2 Experiment Procedure

Figure 5.1 shows the number of faults found by each of the seventy-three inspectors. The number ranges from a minimum of one to a maximum of thirteen, with the majority of inspectors finding either two, three or four faults.

To compare the performance of the estimators using data from a varying number of inspectors as input, virtual inspection teams were created with the team size ranging from 1 to 73. The process of creating virtual inspection teams consisted of randomly selecting the appropriate number of inspectors from the overall pool of 73 inspectors. For example, to create the 17 member inspection teams, 73 inspectors were randomly selected. Then, a matrix of the inspection data from these 17 inspectors was created as described in Figure 1. Using this approach, ten virtual inspection teams were created (with replacement) for each team size, i.e. ten virtual inspection teams of size two, another ten virtual inspection teams of size three, and so on, up to a team size of 73. This process resulted in the creation of a total of 721 inspection teams, 10 for each inspection team size (1- 72) and one team that combines all the 73 inspectors. Automated tools, namely CAPTURE and CARE-2 [96], was used to calculate the estimates.

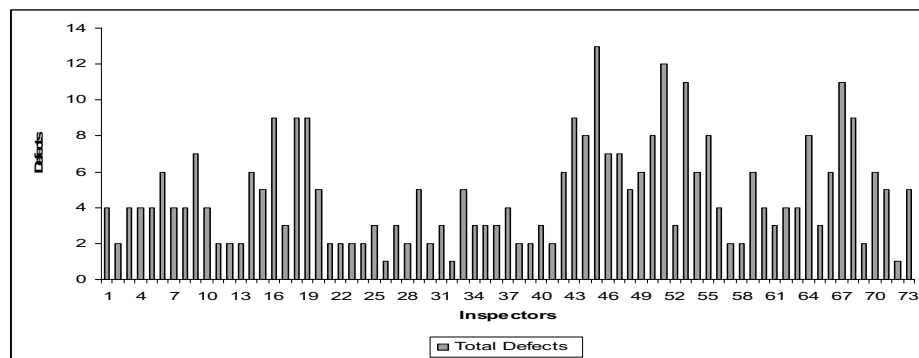


Figure 5.1

Total Fault(s) Found by Each Inspector

5.1.3 Evaluation Criterion

For each input size (1-73), the ten estimates produced are used to compute the median. The estimators are then evaluated on their performance using three parameters: accuracy (bias), precision (variability), and failure rate as described below:

- The **accuracy (bias)** is measured as the relative error (R.E) of an estimate. It is calculated as:

$$\text{Relative error} = (\text{Estimated number of faults} - \text{Actual number of faults}) / \text{Actual number of faults}.$$

A R.E of zero means absolute accuracy (zero bias), a positive R.E. means an overestimation, and a negative R.E means an underestimation. The accuracy of the estimator is measured by calculating the median relative error for each inspection team size. As discussed in Section 2.4.2, the accuracy of an estimate is considered satisfactory when the R.E. is within +/- 20% of the actual value [19, 26-27];

- The **precision** of an estimator is measured by calculating the variability of the then R.E. estimates for each input size (1-73). R.E variability around the central tendency i.e. (median value) is measured using the inter quartile range of the 25th percentile to the 75th percentile.
- The **failure rate** of an estimator is defined as the number of time an estimator fails to produce any result. Because each estimator makes different assumptions about the data and they all operate on the same data matrix, some estimators can fail if the actual data fails to meet some of its basic assumptions.

5.1.4 Analysis and Results

This section provides analysis of the capture-recapture estimates. First, to provide an overview, Figure 5.2 shows the median estimates for each capture-recapture estimators across all team sizes (all the estimates from the same estimator are connected with a line). Recall that the actual number of defects is thirty. From this figure, some general observations are that:

- The estimators tend to underestimate the defect population when the number of inspectors is small ,
- The accuracy of the estimators tends to improve with more inspectors,
- Some estimators improve faster (i.e., obtain accurate results with a smaller number of inspectors) than the other estimators, and
- Some estimators have a more consistent (less variation) improvement than others.

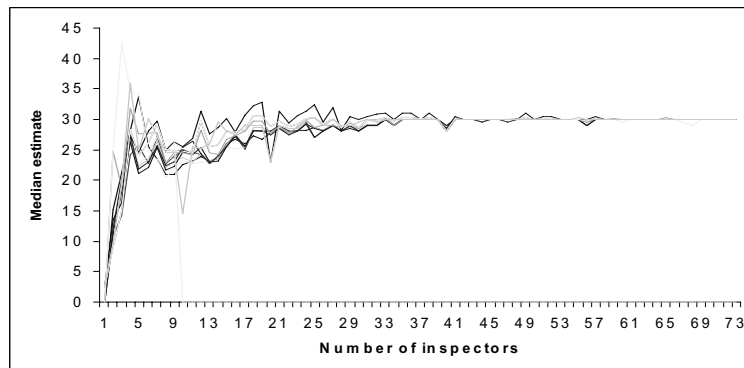


Figure 5.2

Median Estimates as a Function of Number of Inspectors for All CR Estimators

Similarly, the precision values were calculated for each estimator at each team size. Figure 5.3 show the precision of the estimators using the R.E variability among the ten estimates for each team size. Some general observations are that:

- Overall the estimates are highly imprecise when number of inspectors is small,
- The precision of the estimates tend to improve with increasing number of inspectors, and
- The precision of some estimators is more consistent than others.

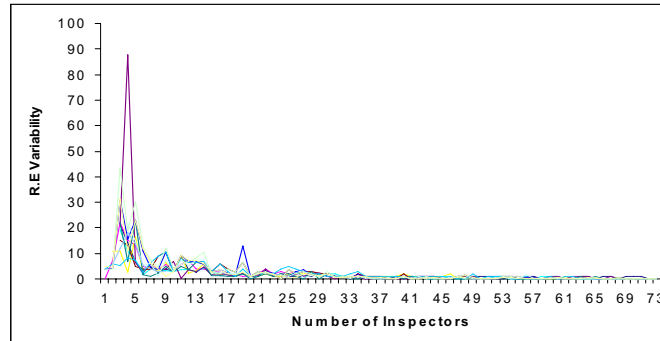


Figure 5.3

Relative Error Variability as a Function of the Number of Inspectors for All CR Estimators

Figure 5.2 and Figure 5.3 indicated that the number of inspectors directly influences the accuracy and precision of the estimators. Analysis of these results produced three regions of the data based on precision, accuracy, and failure rate as follows:

- *Region 1* begins with the estimates based on one inspector. Estimates in this region are both inaccurate and imprecise. Some estimates overestimate while others underestimate. When the median estimates are more accurate, the R.E. variability is still extremely high. In addition, the estimators fail to produce any estimate for some data points in this region.
- *Region 2* begins from where Region 1 ends and includes estimates based on a larger number of inspectors. The estimates in this region are more reliable and show consistent improvement (R.E. variability decreases) as the number of inspectors increases. Both accuracy (median R.E.) and precision (R.E. variability) improve in this region. This region ends at the point where increasing the number of inspectors stops making a significant improvement in the estimate.
- *Region 3* begins at the point beyond which there is no significant improvement in the estimate by increasing inspectors. Therefore, there is little benefit to using more inspectors than the cut-off point that begins the region.

Determining the cut-off points that separate regions 1, 2, and 3 provides insights into the number of inspectors required to generate satisfactory estimates. Determining the cut-off points requires an analysis of the accuracy, precision, and the failure rates of the estimators for each number of inspectors. An approach for combining the analysis of the accuracy and precision of an estimate is to calculate three values for each number of inspectors and estimator combination, namely: a) the median estimate, b) the seventh largest estimate (75th percentile), and c) the third largest estimate (25th percentile). Together b) and c) define the interquartile range.

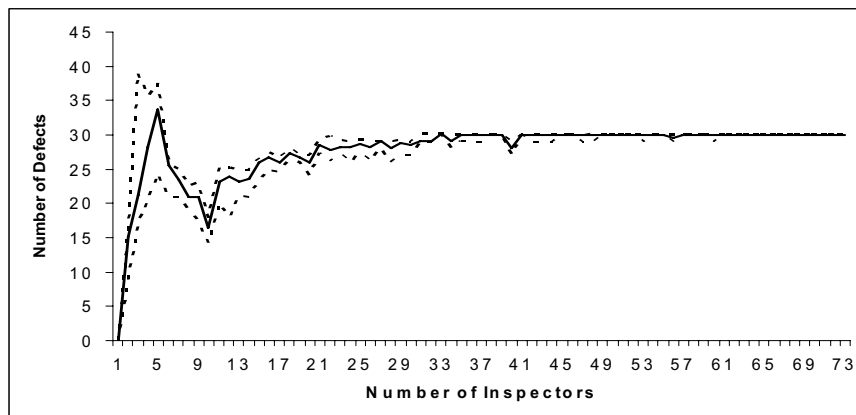


Figure 5.4

Variations of M_0 -CMLE Estimates as a Function of the Number of Inspectors

Figure 5.4 and Figure 5.5 show these three values for two of the estimators with median estimate appearing between the upper (75th percentile) and lower bound (25th percentile) on the estimates. The criterion for selecting the cutoff points for three regions is:

- Cutoff point between region 1 and region 2: Remain in Region 1 as long as all three values (median, 75th percentile, and 25th percentile) have a R.E. greater than 20%. Enter Region 2 at the point when all three values are less than or equal to 20% and they never exceed 20% as the number of inspectors increases.
- Cutoff point between region 2 and region 3: Remain in Region 2 as long as all the three values (median, 75th percentile and 25th percentile) have a R.E. greater than 5%. Cross into Region 3 at the point when all three values are less than or equal to 5% and never exceed 5% as the number of inspectors increases.

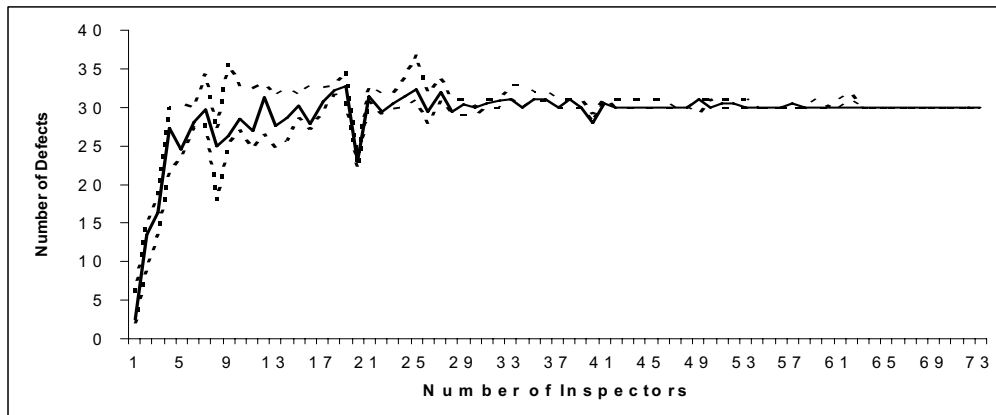


Figure 5.5

Variations of M_h -JK Estimates as a Function of the Number of Inspectors

Table 5.1 shows the cutoff values (number of inspectors) for the three regions using this criterion. For example, the first row of Table 5.1 shows that the M_0 -UMLE estimator stays in Region 1 as long as the number of inspectors is less than or equal to eighteen. When, the number of inspectors is nineteen, the estimator can be relied upon to produce satisfactory estimates and it enters Region 2. It stays in Region 2 and keeps improving as the number of inspectors increases up to forty. When the number of

inspectors reaches forty-one, it crosses over into Region 3 and consistently produces an estimate very close to the actual number of defects.

Since the estimates improve with more inspectors, information about the minimum number of inspectors required to achieve satisfactory estimates is relevant to organizations. The minimum number of inspectors required for achieving estimates with varying levels of accuracy and precision for different estimators can help project managers plan and manage the inspection process. For example, using M_h -SC to produce an estimate with a relative performance of $\pm 20\%$ requires minimum of 15 inspectors, and 41 inspectors to produce an estimate with relative performance of $\pm 5\%$. Therefore, Table 5.2 further details Region 2 and provides the minimum number of inspectors if a project manager wants to achieve precise estimates with a R.E. ranging from $\pm 5\%$ to $\pm 20\%$. The values are calculated using the same criterion as defined earlier.

Table 5.1

CUT OFF POINTS FOR REGIONS 1, 2, AND 3

Estimators	Region 1	Region 2	Region 3
M_o - UMLE	1-17	18-40	41-73
M_o - CMLE	1-15	16-40	41-73
M_o - EE	1-15	16-40	41-73
M_t - UMLE	1-20	21-40	41-73
M_t - CMLE	1-15	21-40	41-73
M_t - EE	1-20	21-40	41-73
M_t -Ch	1-20	21-40	41-73
M_h - SC	1-14	15-40	41-73
M_h - JK	1-25	26-53	54-73
M_h -EE	1-15	16-41	42-73
M_h -Ch	1-14	15-40	41-73
M_{th} -SC	1-14	15-40	41-73

Table 5.2

MINIMUM NUMBER OF INSPECTORS FOR DIFFERENT LEVELS OF PERFORMANCE

Estimators	+/- 5%	+/- 10%	+/- 15%	+/- 20%
M_o- UMLE	41	41	22	18
M_o- CMLE	41	41	23	16
M_o- EE	41	41	21	16
M_t- UMLE	41	41	24	21
M_t- CMLE	41	41	23	21
M_t- EE	41	41	26	21
M_t-Ch	41	41	25	21
M_h- SC	41	41	23	15
M_h- JK	54	54	31	26
M_h-EE	41	41	23	16
M_h- Ch	41	41	23	15
M_{th}-SC	41	41	17	15

5.1.5 *Validity Threats*

- **Threats Addressed:** To reduce the threat to external validity of using students as subjects, the inspectors in this study are Microsoft software professionals. To reduce the threat of using a small number of inspectors, the number of inspectors used in this study was the largest used in any previous study of this type. Also, the inspectors worked independently throughout the inspection process to allow the capture-recapture models to be used.
- **Threats Unaddressed:** While the requirements document inspected in this study was used by students to implement a class project, it may not be representative of an industrial-strength document. Furthermore, the defects were seeded into the document rather than being naturally occurring. But, the defects were seeded into the document for a previous study by researchers who had no knowledge that the results would be used for a capture-recapture study. Therefore, the defects were not seeded in such a way as to specifically benefit a capture-recapture analysis.

5.2 *Study 2: Investigating the Effect of the Number of Faults*

While, Study 1 analyzed the effect of the *number of inspectors* on the accuracy of the CR estimates, this study examined the effect of the other factor, the *number of faults*, on the accuracy of the CR estimates. This analysis was conducted by varying the number of faults while keeping the number of inspectors constant.

This study investigated the effect of the number of faults on the CR estimates using real artifacts developed by students in a senior-level capstone software engineering class (i.e. they were created to guide the later implementation of the system) with naturally occurring defects. The effect of the number of faults on the re-inspection decision is analyzed to gain additional insights into how the CR method can be used on other projects. The major goal of this study was to understand the effect of the number of faults on the estimates produced by CR models. To achieve this goal, this study focused on two important research questions:

Question 1: How does the performance of the CR estimators improve as a larger percentage of faults are discovered?

This question focuses on the general trends in the improvement of the performance of estimators as more faults are found. Answering this question provides details into what percentage of faults must be found before the CR models provide satisfactory estimates. Knowledge of these general trends and analyzing the improvement of the fault count estimates in their organizations will help project manager in determining the quality of an artifact under review.

Question 2: How is the re-inspection decision ability of the CR estimators affected by increasing the number of faults?

This question focuses on the estimate of the post-inspection faults relative to the number of faults found during an inspection without knowing the true faults count. The ability of the CR estimators to accurately estimate the number of post-inspection faults is relevant for making re-inspection decisions. Answering this question provides project managers useful insights to make re-inspection decisions in real development.

5.2.1 Data Set

The data for this study was drawn from the feasibility study [87] (Section 4.3), that evaluated the usefulness of requirement error taxonomy during inspection process:

Table 5.3

REQUIREMENT ARTIFACTS AND INSPECTORS USED IN STUDY 2

Artifact	Name	# of Pages	# of Requirements	Number of Inspectors	Total Defects
A	Starkville theatre system	27	14	8	55
B	Management of apartment and town properties	42	16	8	105

- *Software artifacts and inspectors*: Inspection data from two real requirement artifacts is used in this study. The artifacts were developed by sixteen senior-level undergraduate students, who were enrolled in the Senior Design Course at MSU. The sixteen subjects were divided into two 8-person teams that developed the requirement document for their respective system as shown in Table 5.3. The course required the students to interact with real customers to elicit, and document requirements that they would later implement. Each artifact was then inspected by the same set of developers who created it [87].
- *Software inspection process*. All the subjects performed two inspections of their requirement artifact. The subjects used fault checklist during first inspection and then used the requirement error taxonomy during the second inspection. The same inspection process was followed by the subjects in each

team, and the artifacts were not modified or corrected between inspections (i.e., the same artifact was re-inspected). The total number of faults found after two inspections in each artifact is shown in the last column in Table 5.3.

5.2.2 *Experiment Procedure*

To understand the impact that the number of faults has on the CR estimates, virtual inspections were created by keeping the team size constant at 8 (the number of inspectors) and varying the number of faults from 1 to 55 for artifact A and from 1 to 105 for artifact B. The data from the original study was organized into an 8 (inspectors) X 55 (defects) matrix for artifact A and an 8 (inspectors) X 105 (defects) matrix for artifact B.

To create virtual inspections the appropriate number of rows (equal to the fault count being studied) was selected randomly from the total pool of faults as follows:

- For example, for artifact A, to create the virtual inspection for a fault count of twenty, twenty rows were randomly selected from the 8 X 55 matrix to produce a new 8 X 20 matrix.
- Similarly, for artifact B, to create the virtual inspection for a fault count of thirty, thirty rows were randomly selected from the original 8 X 105 matrix producing a new 8 X 30 matrix.
- Using this approach, 10 virtual inspections (i.e. 10 matrices) were created with replacements for each fault count. i.e., 10 virtual inspection teams are derived from 8 review teams. The automated tools CAPTURE and CARE-2, were then used to calculate the estimates.

5.2.3 *Evaluation Criterion*

From the ten estimates produced for each artifact and each fault count, the median value is calculated. The performance of the CR estimators is then evaluated using three

metrics: accuracy (bias), precision (variability), and failure rate. The definition of these metrics is the same as provided in Section 4.1.2.

5.2.4 Analysis and Results

Figures 5.6 and 5.7 show the median R.E. for each CR estimator for all fault counts (each line connects the estimates from the same estimator) for artifact A and artifact B. To calculate the RE values, the actual number of faults is assumed to be the total number of faults found after two inspections (i.e., 55 for artifact A and 105 faults for artifact B).

These figures show that the estimators severely underestimate when the fault count is small, but the accuracy of estimators improve with a higher fault count. Also, some estimators obtain more accurate results with fewer faults than other estimators. To compare the relative performances of the estimators with respect to fault count, it is

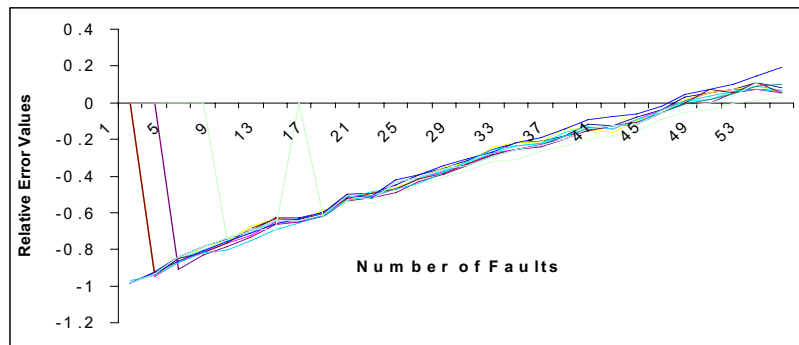


Figure 5.6

Median Relative Error Values for Different Fault Counts for Artifact A

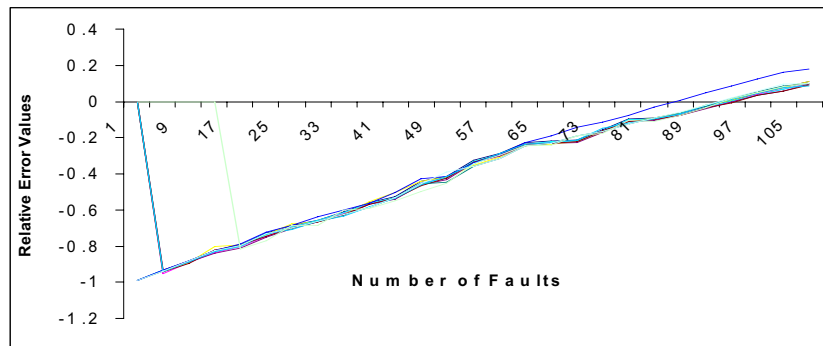


Figure 5.7

Median Relative Error Values for Different Fault Counts for Artifact B

important to determine the cut-off points (minimum fault count) required to obtain an estimate that falls within 20% of the actual value. The cut-off point is one larger than the largest number of faults for which the median estimate falls within 20% (i.e., from that point forward, the R.E. decreases as the fault count increases). For example, for artifact A, for M_0 -EE estimator, from 37 to 55 faults the median estimate is always within 20% of the actual value (i.e., 20% of 55 = 44 faults).

The R.E. in the estimate is only an indicator of accuracy. In practice, an estimator needs to be both accurate and precise. To understand precision, the variability of the R.E. for each fault count is calculated as the size of the interquartile range (i.e., the middle 50% of the data). Then, to combine accuracy and precision, three values are calculated for each fault count: a) the median estimate, b) the seventh largest estimate (75th percentile), and c) the third largest estimate (25th percentile).

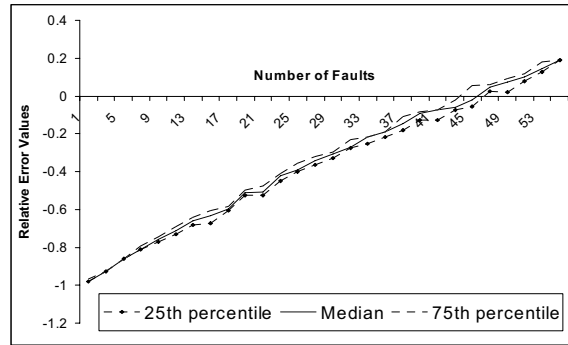


Figure 5.8

Variability in the Estimates for Different Fault Counts for Artifact B

Figure 5.8 shows this analysis for the M_h - JK estimator on artifact A, with the median R.E. estimate appearing between the upper (75th percentile) and lower bound (25th percentile) on the estimates. Similar graphs were produced for each estimator on each artifact. The criterion for selecting the minimum fault count considers the R.E. at three points (median, 75th percentile, and 25th percentile). The result of the analysis that combines accuracy and precision is shown in Table 5.4. Based on these results, some general observations are that:

- For artifact A, depending on the estimator, inspectors need to find anywhere between 62% and 67% of the total faults for the R.E. estimate to be within 20% of the actual value; and between 69% and 82% of total faults for the R.E. estimate to be within 10% of the actual value, and
- Similarly for artifact B, inspectors need to find anywhere between 59% and 69% of total faults for the R.E. estimate to be within 20% range; and between 70% and 79% of total faults for the R.E. estimate to be within 10% range.

Table 5.4

PERCENTAGE OF FAULTS REQUIRED FOR DIFFERENT LEVELS OF
ESTIMATION ACCURACY

Artifact A		Estimator	Artifact B	
-10%	-20%		-20%	-10%
76%	67%	M _o -CMLE	68%	73%
76%	67%	M _o -UMLE	69%	77%
78%	66%	M _o -EE	69%	74%
76%	66%	M _t -CMLE	69%	74%
80%	67%	M _t -UMLE	68%	78%
80%	66%	M _t -EE	69%	78%
80%	66%	M _h -SC	64%	75%
69%	62%	M _h -JK	59%	70%
80%	66%	M _h -EE	68%	74%
80%	67%	M _{th} -SC	64%	75%
82%	69%	M _{th} -EE	68%	79%

Therefore, a significantly large percentage of faults have to be found before the CR models can provide satisfactory estimates. Project managers can perform a similar analysis after each inspection to identify any trend in fault count estimate to gain insights into the quality of an artifact under review. For example, if there is a continuous significant increase in the estimated fault count with increase in the faults found (as opposed to a point after which no improvement is visible), then it is likely that there are substantially more faults remaining in the artifact. The lack of an increase in the estimates indicates that a large percentage of faults have been found.

The above results compared the error in the CR estimates relative to the actual fault count (i.e., total faults found after two inspections). However, during real software development, a project manager does not know the actual fault count. Therefore, they

have to make a re-inspection decision based only on the number of faults found during an inspection and the estimate of the remaining faults.

Therefore, further analysis is performed that re-calculates the R.E. values for each fault count without using the knowledge of the total faults found after two inspections. In this analysis, the actual number of faults is set equal to the particular fault count being studied. For example, for artifact A and fault count equal to 37 (i.e., the fault count being studied), M_h -JK estimator produced a median estimate of 49 faults. Therefore, the R.E. = $(49-37)/37 = 0.32$. Meaning that an estimated 32% more faults are remaining. Figure 5.9 and Figure 5.10 shows the relative estimates of the remaining faults from all the CR estimators at each fault count for artifacts A and B respectively.

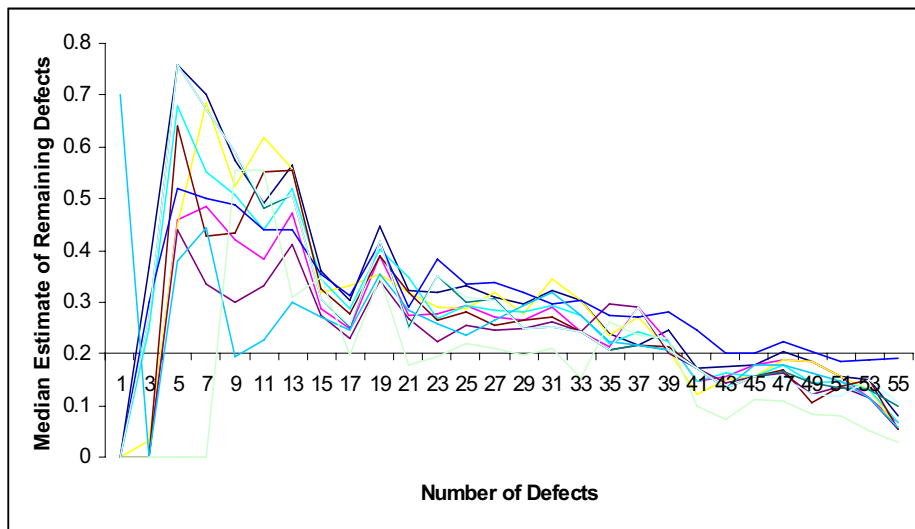


Figure 5.9

Relative Estimate of Remaining Faults for All Counts for Artifact A

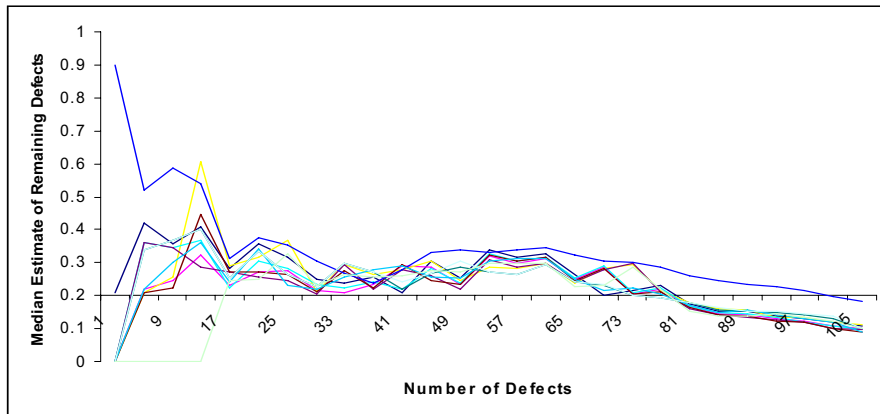


Figure 5.10

Relative Estimate of Remaining Faults for All Counts for Artifact B

To analyze the effect of the fault count on the re-inspection decision, the estimate of remaining faults is compared against 20% criterion (i.e., if the estimate is greater than 20%, a re-inspection should be performed, otherwise no). The general observations from these figures are:

- All the estimators (except EE and UMLE) failed for a fault count of less than five. The EE and UMLE estimators failed for fault counts up to 13,
- For artifact A, the CR estimators (except M_{th} -EE and M_h -JK) correctly suggest the need for re-inspection when the number of faults found are between 5 (9% of actual fault count) and 40 (72% of actual faults). The M_h -JK indicates a need for re-inspection even when the fault count is 49 (i.e., 89% of actual fault count).
- For artifact B, the SC estimator for M_h and M_{th} suggest the need for re-inspections when the fault count is less than 76 (72% of actual fault count). The remaining estimators (except M_h -JK) indicate the need for re-inspection when the fault count is less than 86 (82% of actual fault count). M_h -JK suggests the need for re-inspection when the fault count is less than 98 (93% of actual fault count).

Because the estimator fail when the fault count is less than five, for faults counts greater than five, the correctness of the decision to re-inspect artifacts A and B for all the CR estimators was evaluated. The process for evaluating the correctness is to compare the R.E. in the estimate produced by the CR estimators to the R.E. in the estimate using an ideal estimate (i.e. the CR estimators are perfect). The calculation of these two R.E values is described as:

- R.E in the remaining faults produced by CR estimators; $R.E = (estimate - actual) / actual$; where **actual** = fault count being analyzed and **estimate** = fault count estimated by CR estimators.
- *Ideal R.E in the remaining faults: $R.E = (estimate - actual) / actual$; where **actual** = same as in a) and **estimate** = 55 for artifact A and 105 for artifact B*

For example, for artifact A, R.E in the estimate produced by the M_h -SC estimator at a fault count of 29 = $(36.25 - 29) / 29 = 0.25$ i.e., 25%, and the ideal R.E = $(55 - 29) / 29 = 0.89$ i.e., 89%.

Figure 5.11 shows this analysis graphically for the M_h -SC estimator at all fault counts on artifact A from five to 55 faults. Similarly, Figure 5.12 shows the analysis for M_h SC on artifact B. The solid line represents the R.E in the estimates produced by the M_h -SC estimator and the dotted line represents the ideal R.E. values.

The dotted lines in Figure 5.11 and Figure 5.12 touch the 20% axis at fault count of 45 for artifact A and 86 for artifact B (i.e., after this point the estimates are always within 20% of the actual). Comparing the R.E. in the estimates produced by CR estimators vs. the ideal R.E yields following observations:

- The CR estimators severely underestimate the actual fault count, however, at most fault counts the CR estimators can accurately predict the need for a re-inspection (by predicting that more than 20% of the faults remain).

- Regarding a correct or incorrect re-inspection decision:
 - For artifact A, the estimators correctly suggest the need for re-inspection for 5-40 faults, and the JK estimator makes the correct suggestion for 3-45 faults.
 - For artifact B, estimators correctly suggest the need for re-inspection for 5-76 faults, and JK estimator estimates the need correctly for 3-86 faults.
- However, the JK estimator incorrectly suggests the need for re-inspection for 47-49 faults for artifact A and for 87-98 faults for artifact B.

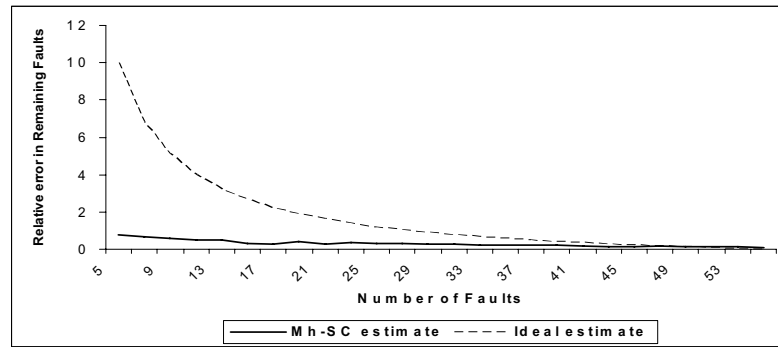


Figure 5.11

Comparisons of Estimate and Actual Relative Error for M_h -SC on Artifact A

This result is shown in Table 5.5. The result show that the Jackknife (JK) estimator is able to accurately predict the need for re-inspection for fault counts up to 44 (in artifact A) and 86 (in artifact B). However, it estimates that there are still some faults remaining for fault count up to 49 faults (in artifact A) and 97 (in fault B). One caveat to this analysis is that we assumed the actual fault count to be equal to the number of faults found after two inspections. There might actually be some more faults in the document. Therefore, we suggest Jackknife as the best estimator.

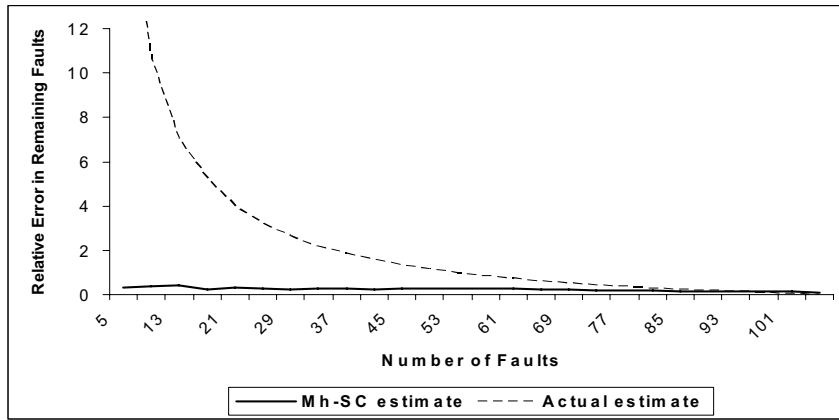


Figure 5.12

Comparisons of Estimate and Actual Relative Error for M_h -SC on Artifact B

Table 5.5

CORRECTNESS OF RE-INSPECTION DECISIONS FOR ARTIFACTS A AND B

Artifact A		Artifact B	
Incorrect decision	Correct decision	Correct decision	Incorrect decision
Less than 5 faults			Less than 5 faults
41-44 faults	7-41 faults	9-76 faults	77-85 faults
47-49 faults for M_h -JK	3-45 faults for M_h -JK	3-86 faults for M_h -JK	86-97 faults for M_h -JK

The overall results from this study provided insights into the: 1) the percentage of faults that have to be found to obtain an accurate CR estimate, and 2) the ability of different CR estimators to correctly predict the need for re-inspection. This section

provides some additional insights into how project managers can use these results to manage projects in their organization.

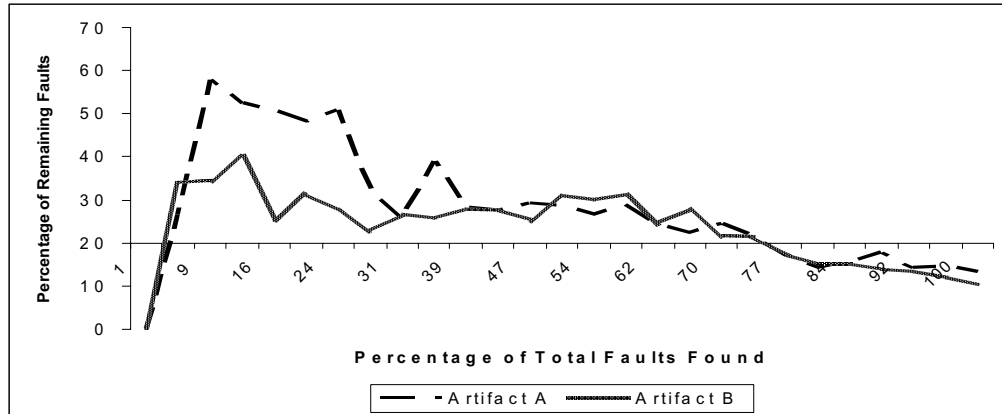


Figure 5.13

Remaining Faults at Different Percentage of Total Faults for Both Artifacts

In real development the number of faults in an artifact is unknown. Therefore, a project manager has to make re-inspection decisions using only the knowledge of the faults that have been found up to that point. To understand the general trend in the estimates from the CR estimators, Figure 5.13 plots the percentage of remaining faults for increasing percentages of total faults for artifact A (solid line) and B (dotted line).

The percentage of remaining faults is calculated as:

$$\frac{[(Estimated\ number\ of\ faults - Number\ of\ faults\ found) / Number\ of\ faults\ found] * 100}{100}$$

Each line connects the median value of the percentage of remaining faults from all the CR estimators (except M_{th} -EE estimator because of its failure rate). Some general observations from Figure 5.13 are:

- The estimators fail to produce an estimate when less than 1% of the faults have been found;
- The estimators consistently (and accurately) predict the need for a re-inspection (i.e., more than 20% of the faults remain) up to the point when approximately 75% of the total faults have been found (this figure is similar for both artifacts).

Therefore, a project manager should allow a re-inspection as long as the estimators predict the need for one. Each organization can perform a similar analysis to ensure these trends are valid. Understanding the trend in the estimates will help project managers decide on the portion of faults that have been discovered and how long the inspection process should continue. However, a project manager has to make a trade-off between the cost involved in re-inspection and benefits of finding more faults.

5.2.5 *Validity Threats*

- **Threats Addressed:** The artifacts used in this study were real software artifacts that were later used to guide implementation of a real system. The faults were naturally occurring and inserted while developing the artifacts rather than being artificially seeded.
- **Threats Unaddressed:** There were some threats to validity that were not addressed. First, the actual number of faults in each document is unknown and might be higher than the assumed fault count (i.e., the total number of faults found after two inspections). The effect of this threat on the results is discussed in next section. A second threat was the artifacts used in this study were developed by student teams in a senior-level capstone course and may not be representative of industrial requirement documents. Also, the nature of faults committed by students may differ from faults made by professionals.

5.3 *Study 3: Evaluating Capture-Recapture Models and Estimators for Estimating the Abundance of Naturally Occurring Faults*

A major difficulty when evaluating CR models is that the number of actual defects is not known beforehand. Researchers have used seeded defects to allow for comparisons of the estimates to the actual value. Accordingly, the findings and recommendations about CR method are based on inspections using artifacts with seeded defects. No empirical study has comprehensively evaluated the CR models in a case where the number of defects is not known beforehand, as would be the case with any real development. Furthermore, software reliability research has shown that seeded, artificial defects differ in detection probability from naturally occurring defects and are easier to detect. Even when re-seeding real defects, their densities differ from that of natural occurring defects. Therefore, it is important to evaluate the CR models in real settings.

Study 3 describes a comprehensive evaluation of CR models and their estimators on real software artifacts that were developed by students in a senior-level capstone software engineering class with naturally occurring defects, and later inspected in the same environment. Each artifact was inspected twice, which allowed the analysis of the CR estimator's ability to make correct recommendations about the need for re-inspection. Therefore, the main goal of this study is to evaluate the CR models and estimators on real software artifacts with naturally-occurring faults, and fault count not known beforehand. More formally, the goal is to:

*Analyze the CR models and estimators
For the purpose of evaluation
With respect to the ability to estimate the number of remaining faults
From the point of view of project managers and inspectors
In the context of a real requirements*

Table 5.6

ARTIFACTS, AND INSPECTORS USED IN STUDY 3

Artifact	Name	Number of Inspectors	1st Inspection Defects	2nd Inspection Defects	Total Defects
A	Starkville theatre system	8	30	25	55
B	Management of apartment and town properties	8	41	64	105
C	Conference management	6	52	42	94
D		6	64	54	118

5.3.1 Data Set

The data for the CR analysis was drawn from Study 1 and Study 3 conducted at MSU that evaluated the requirement error taxonomy (Section 4.3). The information relevant to the CR analysis is provided here:

- *Software Artifacts and Software Inspectors:* Inspection data from four software artifacts is used in this study. The artifacts were developed by senior-level undergraduate students enrolled in the Software Engineering Senior Design Course. So, even though the developers are students, the artifacts are realistic for a small project. The subjects were divided into 4 teams (with 8, 8, 6, and 6 students respectively) that developed the requirement documents for their respective systems as shown in Table 5.6. (Note that though artifact D has the same description as artifact C, it was created by a different set of subjects.) Each artifact was then inspected independently by the same developers who created it.
- *Software Inspection Process:* The inspection process is described in Section 4.3. The same process was used to inspect all four artifacts. Note that the artifacts were not modified or corrected between inspections (i.e. the same

artifact was re-inspected). Therefore, I have inspection data from first inspection, the second inspection, and total for each artifact. Note that the last three columns in Table 5.6 show total unique defects found during the first inspection, unique defects found during the second inspection (different from the defects found during the first inspection), and total defects for both inspections (the sum of the defects from the first and second inspection) respectively.

5.3.2 *Experiment Procedure*

To evaluate the CR models and estimators after each inspection, the estimates for each of the fourteen estimators are calculated as follows:

- *Calculate estimates after first inspection:* For each artifact, the defects found during the first inspection (column 4 of Table 5.6) by all inspectors are inserted into a matrix (as described in Section 2). This matrix is then fed to the automated tools to produce the estimates for all the estimators. Using the estimated defect count and the number of unique defects found at first inspection, the number of remaining defects can be estimated.
- *Calculate estimates after second inspection:* Because the artifacts were unchanged between inspections, to calculate the estimates after second inspection, we use the total defects found from both inspections (column 6 of Table 5.6). It did not make sense to use only the data from inspection 2 because some information would have been excluded making the estimates inaccurate. Therefore, for each artifact, the defects found at the first and second inspection by all inspectors are inserted into a matrix. The matrix is then fed to the automated tools to produce the estimates from all the estimators. Using the estimated defect count and the number of defects found after both inspections, the remaining defects after the second inspection are estimated).

5.3.3 *Analysis and Results*

This section first compares the estimates produced after the first inspection and the second inspection. Second, it analyzes the best CR model. Finally, it discusses how to

use the CR estimators to manage the inspection process. An alpha value of 0.1 is selected in this initial study, because there are only four data points (four artifacts).

5.3.3.1 Comparison of the Estimates After First and Second Inspection

The performance of the CR estimators is compared after the first and second inspection based on the relative error values. Figure 5.14 and Figure 5.15 show the accuracy and precision of the each estimator after the first inspection and after the second inspection respectively. Figure 5.14 and Figure 5.15 partition the relative error values into different regions: the solid line represents absolute accuracy (0 bias), the lower dashed line is a 20% underestimation, and the upper dashed line is a 20% overestimation.

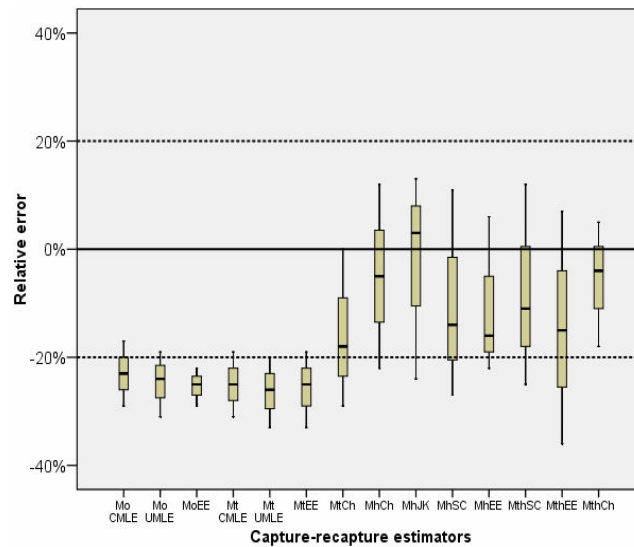


Figure 5.14

Relative Errors in Estimates after First Inspection

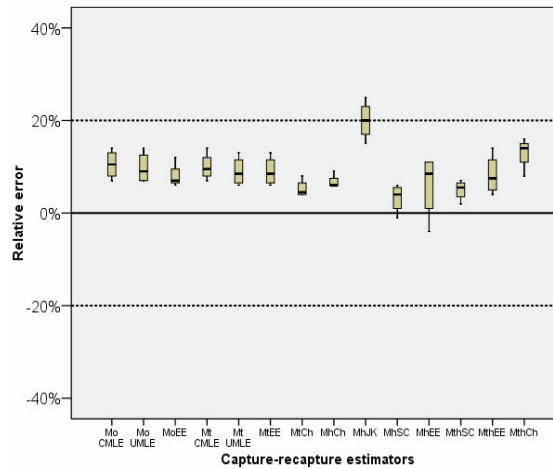


Figure 5.15

Relative Errors in Estimates after Second Inspection

Important observation from Figures 5.14 and 5.15 are:

- In terms of accuracy, there is a general trend that the CR estimators underestimate the defect count after first inspection as most estimators fall below the 0% line and many fall below the -20% region. There is also a general trend that the CR estimators overestimate the defect count after the second inspection, but, most of the estimates (except M_h -JK) fall within the acceptable range of 0%-20%. In addition, the estimators for M_h and M_{th} models are generally less biased than the estimators for the M_o and M_t models at both inspection cycles (except for the M_h -JK estimate after the second inspection).
- In terms of precision, the estimators of M_o and the M_t models are generally more precise (less variation) than estimators from the M_h and the M_{th} models after the first inspection and after the second inspection. Also, the estimators are more accurate and precise after second inspection than after the first inspection.
- In terms of the failure rate of an estimator, M_{th} -EE failed to produce an estimate for artifact C at the first inspection. No other estimator showed any failure.

For each estimator, the relative error after the first inspection was statistically compared with the relative error after the second inspection to determine whether there was any significant improvement. The focus of this analysis was on the magnitude of the relative error and therefore overestimation and underestimation were treated equally. The results from a 2-tailed paired samples t-test show a significant improvement in the accuracy for all the estimators after 2nd inspection over the estimates after 1st inspection.

5.3.3.2 *Selection of the Best Capture Recapture Model*

This study includes four CR models with different estimators for each model: three estimators each for M_o and M_{th} models and four estimators each for M_h and M_t models. This study analyzes the best model(s) to use in software inspections based on the estimates after the first inspection and after the second inspection. The best model(s) were selected using the selection procedure (with a slight modification because of no outliers in our data) originally used by Briand et al., [19], explained as follows:

- First, the best estimator for each model is selected. To choose the best estimator, a 2-tailed paired samples t-test is used to compare the relative error values for the four artifacts (to analyze the direction of improvement and its significance). For each model type, each estimator is compared against every other estimator to select the best estimator for that model. If the t-test does not show significant results in spite of the difference in their mean relative bias values, then the Wilcoxon test is performed (as this test is more powerful under certain situations and can test variability in the estimates). If there is still no statistically best estimator, the estimator with least mean relative error is chosen as the best estimator.
- After selecting the best estimator from each model, the models are compared using 1-tailed t-test to select the best model (major source of variation). We use 1-tailed t-test to test the hypothesis that more sources of variation significantly decreases the bias in the estimate, i.e., M_{th} (with two sources of

variation) is better than M_t and M_h (with one variation source), which in turn are better than M_o (with no variation).

For both statistical tests, an alpha value of 0.1 is selected. This procedure is conducted separately after the first inspection and the second inspection

In these figures, the nodes represent the best estimator selected from each model and the lines represent the relationship between models with the arrow pointing towards the better model. The p-value represents the statistical significance of the improvement (double bold lines indicate significant improvement). The results from this analysis are:

- *Selection of best capture-recapture model after first inspection:* For the M_o model, the results show that CMLE is the best estimator. For the M_t model, the results show that Chao is the best estimator. For the M_h model, the results do not show any difference among the EE, Ch, and JK estimators. I selected the JK estimator because it had the smallest mean relative bias. Finally, Chao was selected as the best estimator for M_{th} type model based on the results of the t-tests. The results from the 1-tailed t-test show that the M_t , M_h , and M_{th} models are an improvement over the M_o model, but this improvement is only significant for the M_h and M_{th} models. Furthermore, the M_{th} model is also a significant improvement over the M_t model, but not a significant improvement over the M_h model. Also, there is no significant difference between the M_h and M_t type models. Based on the number of arrows pointing towards a model, the M_{th} is chosen as the best model for the first inspection.
- *Selection of best capture-recapture model after two inspections:* For the M_o model, the results showed that EE and UMLE are the best estimators. We selected M_o -EE as the best estimator due to its smallest mean relative error. For the M_t model, the results show that Chao is the best estimator. For the M_h model, the results show that SC and Ch are the best estimators. I selected M_h -SC as the best estimator due to its smallest mean relative bias. Again, the M_{th} model results showed that SC and EE are the best estimators. I selected M_{th} -SC as the best estimator because of its smallest mean relative bias. Figure 29, then, shows the results of the selection process for the best model after the second inspection. The results show that the M_t , M_h , and M_{th} models show significant improvement over the M_o model, which is expected (models with any source of variations are expected to be better than model with no source of variation). However, both the M_t and M_h models show a non-significant improvement over the M_{th} model, which was unexpected. Based on the results, the model with two sources of variation (M_{th} model) is no better than

models with one source of variation (M_h and M_t models). On the contrary, the M_t and M_h models showed an improvement over M_{th} model, even though the improvement is not significant. This result also contradicts the earlier finding that the M_{th} model is better than other models when there are a large number of defects and inspectors. Furthermore, there is no significant improvement between the M_t and M_h models. Based on the number of arrows pointing towards a model, the M_h model is chosen as the best model after the second inspection.

The results allow for the following observations:

- A model with two sources of variation do not always show an improvement over models with one source of variation, but always show significant improvement over a model with no sources of variation.
- The models M_h and M_{th} show a significant improvement over the M_0 model, whereas the M_t model does not always show significant improvement over the M_0 model.
- Neither of the models with one source of variation (i.e. M_t and M_h) is significantly better than the other. and
- There are no general trends in the selection of the best estimator for each model type (except Chao estimator for the M_t model).

5.3.3.3 *Using Capture Recapture to Manage the Inspection Process*

Another thrust of this study was to investigate whether project managers can trust CR estimates to manage inspection of software artifacts in real-time. Accordingly, this study analyzed the ability of the CR estimators to provide insight into the quality of the software artifacts. To accurately manage the inspection process, the decision about whether to re-inspect an artifact based on the CR estimates after the first and the second inspection is also evaluated. Since the actual number of defects in the artifacts is unknown, the inspectors' subjective estimates are analyzed to gain insights into the results.

The estimates of defects remaining in an artifact help in deciding when to stop the inspection process. I first compare the estimated remaining defects after first inspection and then after the second inspection with the 20% threshold to make a decision on the need for re-inspection (i.e., if the defects remaining are greater than 20%, a re-inspection is needed). As in actual development, CR models are used to estimate the remaining defects after an inspection using data only from that inspection. Accordingly, we estimate the remaining defects after the first inspection without using the defect information from second inspection. The percentage of remaining defects after an inspection is calculated for each estimator as:

$$\text{Relative estimate of remaining defects} = (\text{Estimated total defects} - \text{defects captured during an inspection}) / \text{Estimated total defects}$$

For example, for artifact A and M_0 -CMLE estimator, the number of defects remaining after first inspection is;

$$\begin{aligned} \text{Remaining defects} &= [39(\text{estimated defects}) - 30 (\text{found})] / 39 (\text{estimate total defects}) \\ &= .23 (\text{i.e., } 23\% \text{ of defects remain}) \end{aligned}$$

The estimates of the remaining defects are calculated relative to each estimator. The estimates from all the 14 estimators are used to compute the median and range of the percentage of remaining defects for each artifact. Figure 5.16 compares the percentage of remaining defects for all artifacts after the first inspection (the dotted line shows the 20% threshold). Figure 5.16 also shows the actual percentage of additional defects found during the second inspection for each artifact to give an indication of the accuracy of the estimates after the first inspection.

After the first inspection, the median estimated remaining defects for all the artifacts is greater than 20%, while some estimators estimated 40% or more remaining defects for artifacts B, C, and D. The estimates indicate the need for a re-inspection of all artifacts. The actual data from the second inspection showing the percentage of the additional defects found during re-inspection (A- 45%, B- 43%, C- 45%, D- 46%) supports this decision. During the original study, the CR models were not used, so the only data which was available to decide on a re-inspection was the subjective opinions of the inspectors. In this case, the subjective opinion of developers supported the recommendation of the CR estimators. The inspectors for each artifact felt that there were defects remaining after first inspection and so a re-inspection was performed.

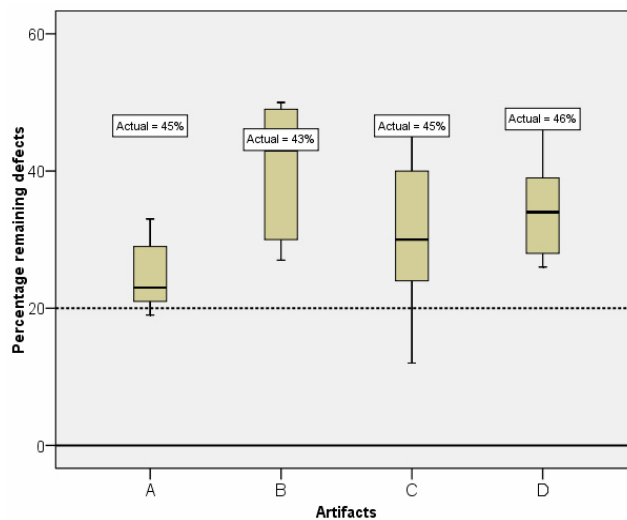


Figure 5.16

Estimated Remaining Faults after the First Inspection

Similarly, the number of remaining defects after the second inspection is estimated relative to each estimator using the defect data from both inspections. For example, for artifact B and M_t -Ch estimator:

$$\begin{aligned} \text{Remaining defects} &= [91(\text{estimated defects}) - 81(\text{found})] / 91 (\text{estimate total defects}) \\ &= .11 \text{ (i.e., 11\% of defects remain)} \end{aligned}$$

The percentage of estimated remaining defects after the second inspection for each artifact is shown in Figure 5.17. After the second inspection, the median estimate of remaining defects as well as extreme outliers for all the artifacts never exceeds the 20% threshold. There is some variation in the estimates for artifacts C and D, but the estimate never exceeds 20%

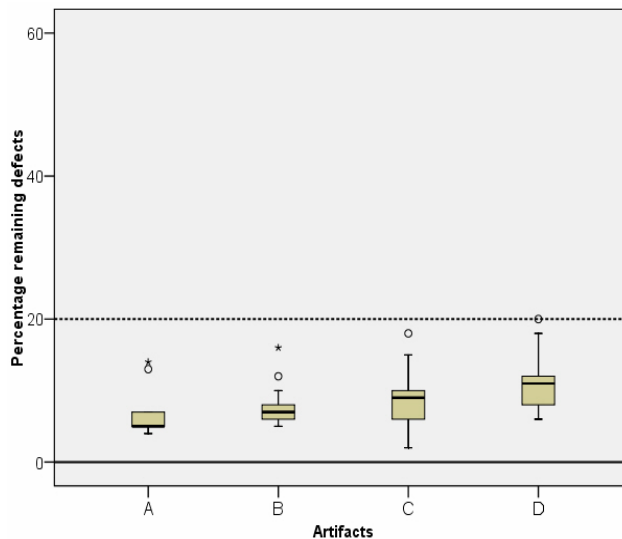


Figure 5.17

Estimated Remaining Faults after the Second Inspection

The results indicate that there is no need for further re-inspections for any of the artifacts; and the inspection process should be stopped. The inspectors' subjective opinion regarding the remaining defects after the second inspection (which was all that was available) supported the recommendation of the CR estimates. The inspectors agreed that they had located all the defects present in the artifact during second inspection, ruling out any need of further inspection. So, the inspection process was stopped. Combining the results from Figure 30 and 31, the CR estimates is helpful to managers for deciding on the need of re-inspection under realistic inspection conditions.

5.3.4 Validity Threats

- **Threats Addressed:** The artifacts used in this study are real software artifacts that were later used to guide implementation. The defects were naturally occurring and inserted while developing the artifacts rather than artificially seeded. The subjects were provided an equal amount of time to perform the first and second inspection, thereby avoiding any bias.
- **Threats Unaddressed:** There were some threats to validity that were not addressed. First, the actual number of defects present in each document is not known and might actually be higher than the assumed defect count (i.e., the total number of defects found after two inspections). This threat especially affects the evaluation of the prediction after the first inspection. A second threat was the artifacts used in this study were developed by student teams in a senior-level capstone course and may not be representative of industrial strength requirement documents. Also, the nature of faults committed by students during development can differ from the faults made by software professionals.

5.4 Discussion of Results

This section summarizes the major findings and recommendations from the three CR studies.

5.4.1 *Summary of Major Findings from Study 1*

The results support the conclusion that increasing the team size directly improves the accuracy, precision, and failure rate of the estimators. The results in Table 5.1 and Table 5.2 provided details of the minimum number of inspectors required for achieving varying levels of estimation performance.

The result from this study suggested that at least 15 to 26 inspectors are needed for achieving a satisfactory estimate. A possible cause of this high number of minimum inspectors is due to the nature of fault data used in this study. Figure 5.1 show that the majority of the inspectors found only two, three, and four faults out of thirty total faults. As a result, the fault data used by capture recapture models in this study was sparse and consequently the the overlap of the faults among inspectors was small.

The data used in Study 3 (as shown in Section 5.3) used inspection data from six and eight inspectors that had larger overlap of faults. The results from this study showed that the estimates produced by the capture recapture models were satisfactory for the inspection team size of six and eight inspectors. Therefore, the distribution of the fault data affects the estimation results.

5.4.2 *Summary of Major Results from Study 2*

The results from this study helped answered two research questions as follows:

- *RQ1: How does the relative performance of the CR estimators improve as a larger percentage of faults are discovered?*

The increase in the number of faults found directly improves the accuracy of estimators with little improvement in precision (or variability). Based on the results, the CR estimators (except M_{th-EE}) can be used on artifacts that contain

five or more faults to produce some estimate. However, the CR estimators require inspectors to find a significantly large percentage of the total faults in order to achieve a satisfactory estimate. Therefore, the CR estimators severely underestimate the actual fault count when a small percentage of the total faults have been found.

- *RQ2: How is the re-inspection recommendation of the CR estimators affected by an increase in the fault count?*

Even though the estimator severely underestimates at different fault counts, it correctly predicts the need for re-inspection very consistently. Based on the results from this study, the CR estimates should not be used for fault counts of less than six for deciding on the need of re-inspections (because of their high rate of failure). In addition, the M_{th} -EE estimator fails to produce an estimate for less than thirteen faults. Therefore, the M_{th} -EE estimator is not recommended.

5.4.3 Summary of Major Findings from Study 3

The results from this study provided insights into the: 1) relative performance of the CR estimates after the first and second inspection, 2) best CR model and estimator, and 3) the ability to make correct reinspection decisions after each inspection cycle:

- *Quality of estimates:* As expected, the number of faults influences the quality of estimates. The estimates are highly negatively biased after first inspection and become positively biased after second inspection as more faults are found. The improvement is statistically significant for all the estimators. Considering the fact that there might be more defects remaining after second inspection, that were not included in the analysis, the estimates may actually be more negatively biased after first inspection and less positively biased after second inspection.
- *The best capture recapture model:* This study tested the hypothesis that:
“A model with two sources of variation (varying inspector abilities and varying defect detection probabilities) is significantly better than models with one source of variation (varying inspector abilities or varying defect detection probability), which in turn are significantly better than model with no variation source”.

Using the estimates from the first inspection alone, the result did not follow this trend. The results are even more unexpected after the second inspection, as there is no significantly better model between the models with one source of variation (M_t and M_h) and the model with two sources of variation (M_{th}). In summary, M_{th} is the best

model after the first inspection whereas M_h is the best model after the second inspection. Therefore, defect detection probability is always a source of variation, while the inclusion of varying inspector ability does not always improve the estimation.

- *Determining quality of a software artifact:* The results show that the CR estimators can help managers accurately decide on the need for re-inspection of an artifact. The results also show that the subjective estimates are similar to objective estimates if the inspectors are same people who developed the requirements because they can provide much better assessment. It is not clear how the subjective estimates would differ if the inspectors were not involved in the artifact development. This is a future research issue that must be investigated.

CHAPTER VI

IMPORTANCE OF RESULTS

This chapter discusses the importance of the results provided in Chapter IV and Chapter V in terms of the overall research hypotheses. This chapter discusses the results with respect to the two hypotheses from Section 1.3.

The first hypothesis of this dissertation was that:

“A taxonomy of requirement errors can be developed and is an effective software approach for improving software quality”

An underlying goal of this research was that addressing software quality problem by focusing on errors will provide better results than focusing only on faults. To accomplish this goal, this dissertation used information about human errors from cognitive psychology to develop the requirement error taxonomy that can be used by software developers to improve the quality of software artifacts.

Based on the evidence gathered from the systematic review (in Chapter III) and the results from the empirical validation of the requirement error taxonomy (in Chapter IV), the requirement error taxonomy was developed and is a significant improvement over fault-based approaches for improving the quality of software artifacts. The results from the empirical studies also provided important insights into the cognitive processes employed by the software engineers and where those processes are likely to fail.

Therefore, this work has positively contributed to the understanding of the software quality assurance through the inclusion of research from cognitive psychology.

The second hypothesis of this dissertation was that:

“The capture-recapture models can support software inspections by estimating the post-inspection defects and predicting the need of re-inspection of software artifacts”.

Another goal of this research was to investigate the use of Capture-Recapture models to manage the software inspection process by accurately estimating the number of defects remaining in an artifact post-inspection and determining whether a re-inspection is necessary.

Based on the results provided in Chapter V, capture-recapture models will help manage the software inspection process and achieve the desired quality of software artifacts. The results in this dissertation also show project managers how to monitor the quality of artifacts by using the estimates of the post-inspection defects from different capture-recapture estimators in real-time development (i.e., when the total number of defects in an artifact is not known beforehand). Therefore, this work has also contributed to support defect size estimates of software artifacts and manage software quality.

The result from the capture recapture analysis also showed that the model that incorporates varying defect detection probability is the best capture recapture model. This capture recapture model calculates the capture frequencies and probabilities of the defects to calculate the estimates. My future work will investigate the cause of the defects with higher capture probabilities by mapping it back to the requirement error taxonomy to

investigate the errors responsible for these defects. The importance of this work will provide insights into the causes of these defects and help narrow the focus on their early detection and removal during software development.

CHAPTER VII

CONCLUSION

This chapter discusses the contribution of this work to the software engineering research and practice community. This chapter also provides a summary of the dissertation work.

7.1 Contribution to Research and Practice Communities

This work brings the knowledge of human errors from cognitive psychology and the knowledge of capture-recapture from wildlife and biology to bear on the problem of software quality. This work will benefit researchers and practitioners in improving the understanding of software quality through inclusion of research from different domains.

For researchers, this work can serve as a starting point for future research into the development of error taxonomies for other lifecycle phases that will provide additional insights into the cognitive aspects of software development and quality assurance. Also, researchers will be motivated to investigate other areas in capture-recapture research that will focus on using the capture-recapture in software organizations and report the results as case studies or surveys.

For practitioners, software engineers will gain a more psychological basis for understanding software faults and errors, and developers will be provided a new set of tools to prevent, detect, and estimate the errors and resulting faults. Future research into the development of concrete techniques based in the error taxonomies will be used by developers to improve software engineering practice industry-wide, benefiting software quality and reducing software development costs. Also, future research into the application of capture-recapture will benefit software project managers and software development organizations.

7.2 Summary

This dissertation work investigated the sources of problems in software artifacts by adapting the concepts of human errors from cognitive psychology to the software engineering domain. This dissertation developed the requirement error taxonomy, and validated the requirement error taxonomy for improving the effectiveness of inspectors during an inspection process. This work provided further methods to support the software inspection process by adapting the concepts of capture-recapture to estimate the post-inspection defects and manage the quality of software artifacts. This dissertation work can be summarized as follows:

- A systematic process of enumeration and classification of the types of errors that can occur during the requirements phase of the software development process;
- An extensive empirical validation of the *Requirement Error Taxonomy* for improving the quality of software artifacts;
- Investigating the use of Capture-Recapture in wildlife research and its use in software engineering domain; and

- An extensive empirical validation of the *Capture-Recapture* method for measuring the quality of software artifacts and managing software inspection process.

7.3 Publications

This section lists the publications that have resulted from this dissertation work:

Journal

- IST Walia, G., and Carver, J. "A Systematic Literature Review to identify a classify Software Requirement Errors." To appear in *Information and Software Technology* (Accepted Date: 29 January 2009).

Refereed Conferences (in reverse chronological order)

- ICST'09 Walia, G., and Carver, J. "Evaluating the Effect of the Number of Naturally Occurring Faults on the Estimates Produced by Capture-Recapture Models," *To Appear in the Proceedings of the 2nd International Conference on Software Testing, Verification, and Validation*. April 1-4. Denver, CO, USA.
- ISSRE'08 Walia, G., and Carver, J. "STUDENT PAPER: The Effect of the Number of Defects on Estimates Produced by Capture-Recapture Models," *To Appear in the Proceedings of the 19th IEEE International Symposium on Software Reliability Engineering*. November 10-14. Seattle, WA, USA.
- ESEM'08 Walia, G., and Carver, J. "Evaluation of Capture-Recapture Models for Estimating the Abundance of Naturally Occurring Defects." *Proceedings of the 2nd International Symposium of Empirical Software Engineering and Measurement*. October 9-10, 2008. in Kaiserslautern, Germany.
- ICSE'08 Walia, G., Carver, J. and Nagappan, N. "The Effect of the Number of Inspectors on the Defect Estimates Produced by Capture-Recapture Models." *Proceedings of the 30th International Conference on Software Engineering*. May 10-18, 2008. Leipzig, Germany.
- ISSRE'07 Walia, G., Carver, J. and Philip, T. "Requirement Error Abstraction and Classification: A Control Group Replicated Study." *Proceedings of the 18th IEEE International Symposium on Software Reliability Engineering*. November 5-9, 2007. Trollhättan, Sweden. p. 71-80.
- ISESE'06 Walia, G. and Carver, J. "Requirements Error Abstraction and Classification: An Empirical Study." *Proceedings of the 2006 International Symposium on Empirical Software Engineering*. Sept. 21-22, 2006. Rio de Janeiro, Brazil. p. 336-345.

Technical Reports (reverse chronological order)

- MSU-081028-2 Walia, G., and Carver, J. "Evaluating the Effect of the Number of Naturally Occurring Faults on the Estimates Produced by Capture-Recapture Models." Technical Report, Department of Computer Science and Engineering, Mississippi State University, August 10, 2008.
- MSU-081028-1 Walia, G., and Carver, J. "Using Error Abstraction and Classification to Improve Quality of Requirements: Conclusions after Three Controlled Experiments." Technical Report, Department of Computer Science and Engineering, Mississippi State University, August 10, 2008.
- MSU-070404 Walia, G., and Carver, J. "Development of Requirement Error Taxonomy as a Quality Improvement Approach: A Systematic Literature Review." Technical Report, Department of Computer Science and Engineering, Mississippi State University, April 4, 2007.

7.3 Future Research Directions

My future research tasks include replicating the procedures performed at Mississippi State University to validate the requirement error taxonomy in a professional development environment. This step will help me understand if the results seen in the classroom are consistent with the results from professional developers. Another future research task is to use the requirement error taxonomy to develop and validate more formalized *defect prevention techniques* (to guide developers during requirement development that will focus their attention on common errors to prevent them from occurring in the first place) and *defect detection techniques* (to help developers detect problems during inspection using different error perspectives). Motivated by the results from this dissertation, I will use a similar approach for developing and validating the error taxonomies for the architecture/design phases of the lifecycle.

I am also interested in researching the psychology of collaborating software developers that have different educational backgrounds during software development to

develop effective software development practices. My future research agenda will also investigate other areas of software engineering through the inclusion of research from cognitive psychology e.g., improving the usability of software development practices, software comprehension, and improving information security.

The application of capture-recapture in software inspection is new and limited to academic context. There are still areas that have not been investigated. Some of the issues that I will explore in the future include:

- Evaluating the applicability of CR estimates on inspection of artifacts that have been updated/corrected in between inspection (as opposed to the successive inspections of same artifact done in Study 3 in Chapter V),
- Combining the CR method with Orthogonal Defect Classification method to analyze the effect of inspectors on estimating the number of defects falling into different defect categories, so the project managers can make more cost-effective decisions on balancing the resources vs. re-inspections, and
- For project managers to recommend the number of inspectors corresponding to different scales of artifact's documentation.

My larger research goals will improve the understanding of software quality assurance as well as research other areas of software engineering through the inclusion of research from cognitive psychology and will improve the application of the CR method for managing the quality of software artifacts.

REFERENCES

- [1] *IEEE Std 610.12-1990*, IEEE standard glossary of software engineering terminology. 1990.
- [2] *Software Engineering Laboratory: Software Measurement Guidebook*. SEL- 94-002. NASA/GSFC Software Engineering Laboratory: 1994.
- [3] A. F. Ackerman, L. S. Buchwald, and F. H. Lewski, "Software Inspections: An Effective Verification Process." *IEEE Software*, vol. 6, no. 3, 1989, pp. 31-36.
- [4] E. B. Allen, "Computer Aided Dispatch System for the London Ambulance Service: Software Requirement Specification," MSU-030429, Department of Computer Science and Engineering, Mississippi State University, 2003.
- [5] V. R. Basili and D. M. Weiss, "Evaluation of a Software Requirements Document by Analysis of Change Data," *Proceedings of the 5th International Conference on Software Engineering*, San Diego, California, 1981, IEEE Press, pp. 314-323.
- [6] Basili, V.R. and Perricone, B.T., "Software Errors and Complexity: An Empirical Investigation," *Communications of the ACM*, vol. 27, no. 1, 1984, pp. 42-52.
- [7] V. R. Basili, and H. D. Rombach, "Tailoring the Software Process to Project Goals and Environments," *Proceedings of 9th International Conference in Software Engineering*, California, 1987, IEEE Press. pp. 345-357.
- [8] D. Batra, "Cognitive Complexity in Data Modeling: Causes and Recommendations." *Requirements Engineering Journal*, 2007. 12(4): 231-244.
- [9] T. E. Bell, and T. A. Thayer, "Software Requirements: Are They Really a Problem?" *Proceedings of Proceedings of 2nd International Conference on Software Engineering*. Los Alamitos, CA: IEEE Computer Society Press. 1976. p. 61-68.

- [10] T. Berling, and T. Thelin, "A Case Study of Reading Techniques in a Software Company," *Proceedings of Proceedings of the 2004 International Symposium on Empirical Software Engineering (ISESE'04)*, IEEE Computer Society, 2004, p. 229-238.
- [11] I. Bhandari, M. Halliday, E. Tarver, D. Brown, J. Chaar, and R. Chillarege, "A Case Study of Software Process Improvement During Development," *IEEE Transactions on Software Engineering*, 1993. 19(12): 1157-1170.
- [12] I. Bhandari, M.J. Halliday, J. Chaar, R. Chillarege, K. Jones, J. S. Atkinson, C. Lepori-Costello, P. Y. Jasper, E. D. Tarver, C. C. Lewis, and M. Yonezawa, "In Process Improvement Through Defect Data Interpretation," *IBM Systems Journal*, 1994. 33(1): 182-214.
- [13] S. Beecham, T. Hall, C. Britton, M. Cottee, and A. Rainer, "Using an Expert Panel to Validate a Requirements Process Improvement Model," *The Journal Of Systems and Software*, 2005, 76(3): 251-275.
- [14] J. Biolchini, P. G. Mian, A C. Natatli, and G. H. Travassos, "Systematic Review in Software Engineering: Relevance and Utility," PESC - COPPE/UFRJ - Brazil: 2005. <http://cronos.cos.ufrj.br/publicacoes/reltec/es67905.pdf>
- [15] B. Boehm, and V. R. Basili, "Software Defect Reduction Top 10 List," *IEEE Computer*, 2001, 34(1): 135-137.
- [16] T. Bove, "Development and Validation of Human Error Management Taxonomy in Air Traffic Control," Ph.D Thesis, Risø National Laboratory & University of Roskilde, 2002
- [17] L. C. Briand, K. E. Emam, B. G. Freimut, and O. Laitenberger, "A Comprehensive Evaluation of Capture Recapture Models for Estimating Software Defect Content," *IEEE Transactions on Software Engineering*, 2000. 26(6): 518-539.
- [18] G. J. Browne, and V. Ramesh, "Improving Information Requirements Determination: A Cognitive Perspective," *Journal of Information and Management*, 2002. 39(8): 625-645.
- [19] P. C. Cacciabue, "A Methodology of Human Factors Analysis for Systems Engineering: Theory and Applications," *IEEE Transactions on System, Man And Cybernetics-Part A: Systems And Humans*, 1997. 27(3): 325-329.
- [20] D. N. Card, "Learning from our mistakes with defect causal analysis," *Software, IEEE*, 1998. 15(1): 56-63.

- [21] R. Chillarege, I. Bhandari, J. Chaar, M. Halliday, D. Moebus, B. Ray, and M. Y. Wong, "Orthogonal Defect Classification- A Concept for In-process Measurement," *IEEE Transactions on Software Engineering*, 1992. 18(11): 943-956.
- [22] J. Coughlan, and D. R. Macredie, "Effective Communication in Requirements Elicitation: A Comparison of Methodologies," *Requirements Engineering Journal*, 2002. 7(2): 47-60.
- [23] W. J. Cristopher, "The Cost of Errors in Software Development: Evidence from Industry," *The Journal of System and Software*, 2002. 62(1): 1-9.
- [24] C. Debou, and A. K. Combelles, "Linking Software Process Improvement to Business Strategies: Experiences from Industry," *Journal of Software Process: Improvement and Practice*, 2000. 5(1): 55-64.
- [25] T. Dyba, "Experiences of Undertaking Systematic Reviews," SINTEF ICT: Queensland.2005.
- [26] S. Eick, C. Loader, M. Long, L. Votta, and S. V. Weil, "Estimating Software Fault Content before Coding," In *Proceedings of the 14th International Conference on Software Engineering*, 1992, Melbourne, Australia: ACM Press: 59-65
- [27] S. Eick, C. Loader, S. V. Weil, and L. Votta, "How Many Errors Remain in a Software Design after Inspection," In *Proceedings of the 25th Symposium on the Interface*. 1993
- [28] K. El-Emam, and O. Laitenberger, "Evaluating Capture-Recapture Models with Two Inspectors," *IEEE Transactions on Software Engineering*, 2001, 27(9): 851-864.
- [29] K. El-Emam, O. Laitenberger, and T. Harbrich, "The Application of Subjective Estimates of Effectiveness to Controlling Software Inspections," *Journal of Systems and Software*, 2000, 54(2): 119-136.
- [30] A. Endres, "An Analysis of Errors and Their Causes in System Programs," *IEEE Transactions on Software Engineering*, 1975, 1(2): 140-149.
- [31] A. Endres, and D. Rombach, "A Handbook of Software and Systems Engineering," 1st ed. 2003, Harlow, England: Pearson Addison Wesley.

- [32] M. R. Endsley, "Situation Awareness and Human Error: Designing to Support Human Performance," In *Proceedings of Proceedings of the High Consequence Systems Surety Conference*, Albuquerque, NM: SA Technologies. 1999. p. 2-9.
- [33] P. M. Fitts, and R. E. Jones, "Analysis of Factors Contributing to 460 'Pilot Error' experiences in Operating Aircrafts Control," In *Proceedings of Selected Papers on Human Factors in the Design and Use of Control Systems*, New York: Dover Publications Inc, 1961. p. 332-358.
- [34] W. A. Florac, "Software Quality Measurement: A Framework for Counting Problems and Defects," CMU/SEI-92-TR-22. Carnegie Mellon Software Engineering Institute: Pittsburgh, PA.1992. <http://citeseer.ist.psu.edu/florac92software.html>.
- [35] B. Freimut, C. Denger, and M. Ketterer, "An Industrial Case Study of Implementing and Validating Defect Classification for Process Improvement and Quality Management," In *Proceedings of Proceedings of the 11th IEEE International Software Metrics Symposium*, IEEE Press, 2005.
- [36] D. A. Gaitros, "Common Errors in Large Software Development Projects," *The Journal of Defense Software Engineering*, 2004, 12(6): 21-25.
- [37] J. Galliers, S. Minocha, and A. Sutcliffe, "A Causal Model of Human Error for Safety Critical User Interface Design," *ACM Transactions on Computer-Human Interaction*, 1998, 5(3): 756-769.
- [38] R. B. Grady, "Software Failure Analysis for High-Return Process Improvement," *Hewlett-Packard Journal*, 1996. 47(4): 15-24.
- [39] T. Hall, S. Beecham, and A. Rainer, "Requirement Problems in Twelve Software Companies: An Empirical Analysis," *IEE Proceedings Software*, 2002, 149(5): 153-160.
- [40] J. H. Hayes, "Building a Requirement Fault Taxonomy: Experiences from a NASA Verification and Validation Research Project," In *Proceedings of Proceedings of the 14th International Symposium on Software Reliability Engineering*: IEEE Computer Society, 17-20 Nov., 2003, p. 49- 59.
- [41] A. Issac, S. T. Shorrock, R. Kennedy, B. Kirwan, H. Andersen, and T. Bove, "The Human Error in ATM Technique (HERA-JANUS)," HRS/HSP-002-REP-03. European Air Traffic Management: 2002.

- [42] J. Jacobs, J. V. Moll, P. Krause, R. Kusters, J. Trienekens, and A. Brombacher, "Exploring Defect Causes in Products Developed by Virtual Teams," *Journal of Information and Software Technology*, 2005. 47(6): 399-410.
- [43] S. H. Kan, V. R. Basili, and L. N. Shapiro, "Software Quality: An Overview From The Perspective Of Total Quality Management," *IBM Systems Journal*, 1994. 33(1): 4-19.
- [44] B. Kitchenham, "Procedures for Performing Systematic Reviews," Technical Report TR/SE-0401, Department of Computer Science, Keele University and National ICT, Australia, Ltd.: 2004. http://www.elsevier.com/framework_products_promis_misc/inf-systrev.pdf
- [45] B. Kitchenham, E. Mendes, and G. H. Travassos, "A Systematic Review of Cross-vs. Within-Company Cost Estimation Studies," In *Proceedings of 10th International Conference on Evaluation and Assessment in Software Engineering (EASE'06)*, Keele University, 2006.
- [46] A. Ko, and B. Myers, "Development and Evaluation of a Model of Programming Errors," *Proceedings of the 2003 Symposia on Human Centric Computing Languages and Environments (HCC'03)*, Auckland, New Zealand, Oct. 28-31. p. 1-8.
- [47] A. Ko, and B. Myers, "A framework and methodology for studying the causes of software errors in programming systems," *Journal of Visual Languages & Computing*, Vol. 16 (2005), p. 41-84.
- [48] J. Kramer, A. L. Wolf, Proceedings of the 8th International Workshop on Software Specification and Design," *ACM SIGSOFT Software Engineering Notes*, Vol. 21, no. 5, Sept. 1996, pp.21-35.
- [49] F. Lanubile, F. Shull, F., V. R. Basili, "Experimenting with error abstraction in requirements documents," In *Proceedings of Fifth International Software Metrics Symposium, METRICS98*, 1998, p. 114-121.
- [50] C. P. Lawrence, and I. Kosuke, "Design Error Classification and Knowledge Management," *Journal of Knowledge Management Practice*, 2004. 10(9): 72-81.
- [51] M. Leszak, D. E. Perry, and D. Stoll, "A Case Study in Root Cause Defect Analysis," In *Proceedings of the 22nd International Conference on Software Engineering*, Limerick, Ireland ACM Press, 2000, p. 428 – 437.
- [52] R. R. Lutz, "Analyzing Software Requirements Errors in Safety-Critical, Embedded Systems," In *Proceedings of the IEEE International Symposium on*

Requirements Engineering. San Diego, CA, USA: IEEE Computer Society Press. 4-6 Jan., 1993. p. 126-133.

- [53] R. R. Lutz, "Targeting Safety-Related Errors During Software Requirements Analysis," *The Journal of systems and software* 1996. 34(3): 223-230
- [54] C. Masuck, "Incorporating A Fault Categorization and Analysis Process in the Software Build Cycle," *Journal of Computing Sciences in Colleges* 2005, 20(5): 239 – 248.
- [55] R. G. Mays, C. L. Jones, G. J. Holloway, and D. P. Studinski, "Experiences with Defect Prevention," *IBM Systems Journal*, 1990, 29(1): 4 – 32.
- [56] T. Nakashima, M. Oyama, H. Hisada, and N. Ishii, "Analysis of Software Bug Causes and Its Prevention," *Journal of Information and Software Technology*, 1999, 41(15): 1059-1068.
- [57] D. Norman, *The Psychology of Every Day Things*. 1988, New York: Basic Books.
- [58] D. A. Norman, "Steps Towards a Cognitive Engineering: Design Rules Based on Analyses of Human Error," *Communications of the ACM*, 1981. 26(4): 254-258.
- [59] D. A. Norman, "Design Rules Based on Analyses of Human Error," *Communications of the ACM*, 1983, 26(4): 254-258.
- [60] K. M. Oliveira, F. Zlot, A. R. Rocha, G. H. Travassos, C. Galotta, and C. S. Menezes, "Domain-Oriented Software Development Environment," *The Journal Of Systems and Software*, 2004, 72(2): 145-161.
- [61] F. Paterno, and C. Santoro, "Preventing User Errors by Systematic Analysis of Deviations from the System Task Model," *International Journal of Human-Computer Studies*, 2002, 56(2): 225-245.
- [62] H. Petersson, T. Thelin, P. Runeson, and C. Wohlin, "Capture-Recapture in Software Inspections after 10 Years Research - Theory, Evaluation and Application," *Journal of Systems and Software*, 2003.
- [63] A. A. Porter, L. G. Votta, and V. R. Basili, "Comparing Detection Methods for Software Requirements Inspections: A Replicated Experiment," *IEEE Transactions on Software Engineering*, 1995, 21(6): 563-575.
- [64] S. L. Pfleeger, and J. M. Atlee, "*Software Engineering Theory and Practice*," 3rd ed. 2006, Upper Saddle River, NJ: Prentice Hall.

- [65] J. Rasmussen, "Human Errors: A Taxonomy for Describing Human Malfunction in Industrial Installations," *Journal of Occupational Accidents*, 1982, 4: 311-335.
- [66] J. Rasmussen, "Skills, Rules, Knowledge: Signals, Signs and Symbols and Other Distinctions in Human Performance Models," *IEEE Transactions: Systems, Man, & Cybernetics*, 1983, SMC-13: 257-267.
- [67] J. Reason, *Human Error*, 1990, New York: Cambridge University Press.
- [68] K. Sasao, and J. Reason, "Team Errors: Definition and Taxonomy," *Journal of Reliability Engineering and System Safety*, 1999, 65(1): 1-9.
- [69] C. B. Seaman, and V. R. Basili, "An Empirical Study of Communication in Code Inspections," *Proceedings of International Conference in Software Engineering*, pp.96-106, Boston, Mass., May 1997.
- [70] C. B. Seaman, "Qualitative Methods in Empirical Studies of Software Engineering," *IEEE Transactions of Software Engineering*, 1999, 25(4), 557-572.
- [71] J. Smith, "The 40 Root Causes of Troubled IT Projects," *Journal of IEEE Computer and Control Engineering*, 2002, 13(3): 109-112.
- [72] G. M. Schneider, J. Martin, and W. T. Tsai., "An Experimental Study of Fault Detection in User Requirements Documents," *ACM Transactions on Software Engineering and Methodology*, 1992, 1(2): 188-204.
- [73] S. T. Shorrock, and B. Kirwan, "Development and Application of a Human Error Identification Tool for Air Traffic Control," *Journal of Applied Ergonomics*, 2002, 33(4): 319-336.
- [74] I. Sommerville, *Software Engineering*. 8th ed. 2007, Harlow, England: Addison Wesley.
- [75] F. Shull, I. Rus, and V. R. Basili, "How Perspective Based Reading Can Improve Requirement Inspection," *IEEE Computer*, 2000, **33**(7): 73-79.
- [76] F. Shull, J. Carver, and G. Travassos, "An Empirical Methodology for Introducing Software Processes," In *Proceedings of Joint 8th European Software Engineering Conference and 9th ACM SIGSOFT Foundations of Software Engineering*, 2001, Vienna, Austria: 288-296.
- [77] N. A. Stanton, and S. V. Stevenage, "Learning to Predict Human Error: Issues of Acceptability, Reliability and Validity," *Journal of Ergonomics*, 1998, 41(11): 1737-1756.

- [78] M. A. Stutzke, and C. S. Smidts, "A Stochastic Model of Fault Introduction & Removal During Software Development," *IEEE Transactions on Reliability*, 2001, 50(20): 184-193.
- [79] A. Sutcliffe, A. Economou, and P. Markis, "Tracing Requirements Errors to Problems in the Requirements Engineering Process," *Requirements Engineering Journal*, 1999, 4(3): 134-151.
- [80] A. Sutcliffe, B. Gault, and N. Maiden, "ISRE: Immersive Scenario-Based Requirements Engineering with Virtual Prototypes," *Requirements Engineering Journal*, 2004, 10(1): 95-111.
- [81] A. Swain, and H. Guttman, "*Handbook of Human Reliability Analysis with Emphasis on Nuclear Power Plant Applications*," Nuclear Regulatory Commission, Washington, DC.1983.
- [82] T. Thelin, P. Petersson, and P. Runeson, "Confidence Intervals for Capture-Recapture Estimations in Software Inspections," *Journal of Information and Software Technology*, 2002, 44(12): 683-702.
- [83] T. Thelin, P. Runeson, C. Wohlin, T. Olsson, and C. Andersson, "Team Based Fault Content Estimation in the Software Inspection Process," In *Proceedings of the 26th International Conference on Software Engineering (ICSE'04)*, IEEE Computer Society, 23-28 May 2004, p. 263- 272.
- [84] C. Trevor, S. Jim, C. Judith, and K. Brain, "*Human Error in Software Generation Process*," University of Technology, Loughborough, England 1994. www.branchlines.org.uk/Research/Tread1.pdf
- [85] S. Viller, J. Bowers, and T. Rodden, "*Human Factors in Requirement Engineering: A Survey of Human Sciences Literature Relevant to the Improvement of Dependable Systems Development Processes*," Cooperative Systems Engineering Group Technical Report, CSEG/8/1997, Computing Department, Lancaster University: Lancaster.1997. citeseer.ist.psu.edu/viller97human.html
- [86] G. S. Walia, "Empirical Validation of Requirement Error Abstraction and Classification: A Multidisciplinary Approach," M.S. Thesis, Mississippi State University, 2006
- [87] G. S. Walia, J. Carver, and T. Philip, "Requirement Error Abstraction and Classification: An Empirical Study," In *Proceedings of IEEE Symposium on Empirical Software Engineering*, Brazil: ACM Press, 2006, p. 336-345.

- [88] G. S. Walia, and J. Carver, "Development of a Requirement Error Taxonomy as a Quality Improvement Approach: A Systematic Literature Review," MSU-070404, Department of Computer Science and Engineering, Mississippi State University: 2007. http://www.cse.msstate.edu/PUBLICATIONS/TECHNICAL_REPORTS/MSU-070404.pdf
- [89] G. S. Walia, J. Carver, and T. Philip, "Requirement Error Abstraction and Classification: A Control Group Replicated Study," *18th IEEE International Symposium on Software Reliability Engineering*, 2007: Trollhättan, Sweden.
- [90] G. S. Walia, J. Carver, and N. Nagappan, "The Effect of the Number of Inspectors on the Defect Estimates Produced by Capture-Recapture Models," *Proceedings of the 30th International Conference on Software Engineering*, May 10-18, 2008, Leipzig, Germany.
- [91] G. S. Walia, and J. Carver, "Evaluation of Capture-Recapture Models for Estimating the Abundance of Naturally Occurring Defects," *Proceedings of the 2nd International Symposium of Empirical Software Engineering and Measurement*, October 9-10, 2008, Kaiserslautern, Germany.
- [92] G. S. Walia, and J. Carver, "STUDENT PAPER: The Effect of the Number of Defects on Estimates Produced by Capture-Recapture Models," *Proceedings of the 19th IEEE International Symposium on Software Reliability Engineering*, November 10-14, 2008, Seattle, WA, USA.
- [93] G. S. Walia, and J. Carver, "Evaluating the Effect of the Number of Naturally Occurring Faults on the Estimates Produced by Capture-Recapture Models," *Proceedings of the 2nd International Conference on Software Testing, Verification, and Validation*, April 1-4, 2009, Denver, CO, USA.
- [94] G. S. Walia, and J. Carver, "A Systematic Literature Review to identify a classify Software Requirement Errors," *Journal of Information and Software Technology*, 2009.
- [95] S. V. Weil, and L. Votta, "Assessing Software Designs Using Capture-Recapture Methods," *IEEE Transactions on Software Engineering*, 1993, **19**(11): 1045-1054.
- [96] G. C. White, D. R. Anderson, K. P. Burnham, and D. Otis, "Capture-Recapture and Removal Methods for Sampling Closed Populations," Los Alamos National Laboratory, 1982.
- [97] X. Zhang, and H. Pham, "An Analysis of Factors Affecting Software Reliability," *The Journal Of Systems and Software*, 2000, **50**(1): 43-56.