EVALUATING THE USEFULNESS OF REQUIREMENT ERROR TAXONOMY AS A DEFECT PREVENTION TECHNIQUE: AN EMPIRICAL INVESTIGATION

A Paper
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

By

Sana Rehman

In Partial Fulfillment
for the Degree of
MASTER OF SCIENCE

Major Department:
Computer Science

April 2014
Fargo, North Dakota

North Dakota State University
Graduate School

Title

Evaluating the Usefulness of Requirement Error Taxonomy as a Defect Prevention Technique: An Empirical Investigation

By

SANA REHMAN

The Supervisory Committee certifies that this ***disquisition*** complies with North Dakota State University's regulations and meets the accepted standards for the degree of

MASTER OF SCIENCE

SUPERVISORY COMMITTEE:

Dr. Gursimran Walia
Chair

Dr. Kendall E. Nygard

Dr. Maria Alfonseca-Cubero

Approved by Department Chair: Dr. Brian M Slator

04/11/2014
Date

Brian M Slator
Signature

ABSTRACT

Defect prevention techniques can be used during the creation of software artifacts to help developers create high-quality artifacts. The Requirement Error Taxonomy developed by Walia et al. [22, 23] helps focus developer's attention on common errors that can occur during requirements engineering. This paper investigates the usefulness of the Requirement Error Taxonomy as a defect prevention technique. The goal was to determine if making requirements engineers' familiar with the Requirement Error Taxonomy would reduce the likelihood that they commit errors while developing a requirements document. We conducted an empirical study in which the participants used the Requirement Error Taxonomy during inspection of a requirements document. Then, in teams, they developed their own requirements document which was evaluated by other students. The hypothesis was that participants who find more errors during the inspection of a requirements document would make fewer errors when creating their own requirements document. The overall result supports this hypothesis.

ACKNOWLEDGEMENTS

I would like to thank all the individuals and organizations who have me accomplishing this task. First of all, I would like to extend my gratitude towards my adviser Dr. Gursimran Walia for his continuous help, support, patience and guidance in the development and completion of this paper. He has been a helpful advisor who has motivated me at every step and guided me during my master's program.

I am grateful to Dr. Kendall Nygard who provided me the opportunity to become part of NDSU and for being a great mentor throughout. His presence on this committee is very special for me and I want to thank him for his time.

I am thankful to Dr.Maria Alfonseca-Cubero from the department of Mathematics who kindly consented to be on my supervisory committee and supported me through this effort.

My journey through the Master's program went through many moments of success and failures but I was always reminded of the famous quote from Winston Churchill who once said:

"Success is not final, failure is not fatal: it is the courage to continue that counts."

I continued through this journey but it was only possible because of the support and guidance of the faculty and staff of Computer Science Department, my family and friends who stood by me as a rock and above all the Almighty.

DEDICATION

I dedicate this research paper to my family who supported me through thick and thin from thousand miles in India and my friends who are my family in the US.

To my parents Dr.H.R.Khan and Dr. (Mrs.) Shamim Khan for having faith and believing in my dreams. Your push for tenacity still ring in my ears and it is only because of your sacrifice and hardwork that I could achieve this milestone in my life.

My brother Hasan Raza Khan and sister Sanobar Khan for their unconditional love, support and for always being by my side. You both are very special.

A very special thanks to my friend and colleague Anshuman Manori who always stood by me in difficult times and inspired me to pursue higher studies from NDSU, USA.I appreciate his help in proofreading this paper.

I am in debt to Dr.Robert Roach who performed the lifesaving brain surgery on me when I was diagnosed with brain tumor. Without his timely diagnosis and treatment I would not have seen this day.

Dr.Gursimran Walia has been the ideal advisor. His sage advice, insightful criticisms and patient encouragement aided me in research in innumerable ways.

Most of all thanks to God for being so merciful by showering his blessings on me and making this possible.

TABLE OF CONTENTS

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

1. INTRODUCTION

Software Engineers are constantly focused on developing high quality software. To address the problem of poor software quality, researchers have devoted considerable effort to developing methods that help developers find and fix problems early when these repairs are easiest and cheapest. Most of the early-lifecycle quality improvement methods focus on faults i.e., mistakes recorded in a requirement or design document. The use of fault detection methods (based on *fault classification taxonomies*) has been empirically evaluated through controlled experiments and case studies in both laboratory and real setting (e.g., [2, 5, 6, 21]). Even when faithfully applying empirically-validated fault-based techniques, developers do not find all problems in the software. As a result, an estimated 40-50% of development effort is spent fixing problems that should have been fixed in an earlier phase or should have been prevented altogether [3]. Therefore, there is a need to improve early defect detection and to help developers eliminate the unnecessary rework.

Because there are a number of competing definitions in the literature, we will first define the terms "*error*" and "*fault*" to avoid any confusion. An *error* is a defect in the human thought process while trying to understand information, solve problems, or use methods and tools. A *fault* is a concrete manifestation of an error within a software artifact. One error may cause several faults and various errors may cause the same fault. The definition of "error" used in this paper more closely resembles the definition of *human error* than that of *program error* (or incorrect program condition) in IEEE standard 610 [1].

The main drawback of software quality approaches that focuses exclusively on faults is that the underlying cause of the fault (i.e., the error) is neither addressed nor identified. An error taxonomy can help developers detect and eliminate errors and related faults. Furthermore, by

identifying errors, developers can find additional related faults that may have been overlooked (similar to a doctor finding and treating all symptoms once he knows the underlying disease). Therefore, an error-based quality improvement approach is needed.

The idea of using error information to improve software quality is not novel. Researchers have used information about source of faults in different ways. Some techniques that focus on errors determine the cause of only a sample of previous faults to suggest software process changes and prevent future faults [4, 7-8, 12, 14-17]. In the cases where techniques do address the underlying cause of faults (e.g., *Root-Cause Analysis [14]*, *Orthogonal Defect Classification [5]*, and *Error Abstraction [12]*), the research has focused primarily on errors from the software engineering domain. These approaches lack a strong cognitive theory to describe the types of mistakes make when creating software artifacts. *Human Error* research in cognitive psychology builds upon theoretical models of human reasoning, planning, and problem solving, and how these ordinary psychological processes fail [11, 13, 18-20]. The exploitation of *human error* research broadens our understanding of errors that software engineers make during development.

To address this issue, we combined information from software engineering and cognitive psychology to develop requirements error taxonomy [24]. We have also evaluated the usefulness and completeness of the taxonomy with a family of four controlled empirical studies [23-26]. Section II provides a brief description of the error taxonomy along with its development and evaluation processes.

The results from our previous studies show that the requirement error taxonomy improves the defect detection effectiveness of both individual inspectors and teams. A second important value of the requirement error taxonomy is that it can focus developers' attention on common errors during the requirement engineering process. An awareness of these common errors should

make developers less likely to commit them and more likely to create an artifact that will have fewer defects to remove during the review and testing.

This paper presents an empirical study to investigate the usefulness of the requirement error taxonomy as a defect prevention technique. A controlled study with university students was performed to determine if students avoided making the errors and faults in their own document that they had found in earlier inspection of someone else's document.

Section II provides some background on the error abstraction process and the requirement error taxonomy. Section III describes the study design. Section IV describes the data analysis results. Section V discusses the threats to validity. Section VI focuses on the relevance of the results. Section VII summarizes the results from this study. Section VIII concludes the paper and presents ideas for future work.

2. BACKGROUND AND RELATED WORK

Our literature review identified nine methods that used causal analysis to determine the source of a fault and suggest preventive actions (e.g., [4, 16]) or process changes (e.g., [7-9, 15, 17]). These method were successful relative to their goals, but were incomplete because they focused on a representative sample of faults (potentially overlooking many errors). Nevertheless, the insights provided by these methods provided input to the requirement error taxonomy. A complete discussion of these methods, their limitations and their contributions to the requirement error taxonomy has been published in a systematic literature review [24].

Previously, other researchers have developed quality improvement approaches based on the source of faults. While these approaches have been effective, they have two shortcomings: 1) they typically do not define formal error identification and repair process; and 2) 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 approaches that were developed based on the sources of faults. Section 2.2 provides an overview of how cognitive psychology research can help identify the sources of faults. Section 3then provides our background work done in developing and validating the requirement error taxonomy.

A discussion about software quality necessarily focuses on a three important terms, which we have referred to: *error*, *fault*, and *failure*. These terms have competing and contradictory definitions in the literature. To reduce confusion, we provide a definition for each term that is used throughout this proposal. These definitions are similar to those provided by Lanubile, et al. [46][46] and are consistent with software engineering (SE) textbooks [27, 60, 74] and an IEEE standard [1].

- An ***error*** occurs when a developer's thought process is flawed. The error may affect one or more elements in a software artifact. Errors may arise from different sources, including such things as *slips* that occur due to incorrect execution of a planned action (e.g. out of order sequence of steps) or *mistakes* that occur due to inaccurate or incomplete understanding of a system (e.g. some system-specific information was misunderstood leading to the selection of wrong method). The term *error* has multiple definitions. In fact, IEEE Standard 610 provides four definitions ranging from an incorrect program condition (referred to as a *program error*) to a mistake in the human thought process (referred to as a *human error*). The definition of *error* used in this proposal more closely resembles the definition of a *human error* rather than a *program error*.
- A ***fault*** is a concrete manifestation of an error, i.e. something written incorrectly in an artifact.
- A ***failure*** is the incorrect execution of software, e.g. a software crash or an incorrect output.

A medical analogy may help illuminate the differences between faults and errors. A patient who is sick with a disease exhibits symptoms, or visible manifestations of that disease. For example, a patient may complain of severe headaches. A doctor can prescribe painkillers for the headaches, but symptomatic treatment does not affect the underlying symptomatic cause. The patient might have migraines and need a beta-blocker or they might have a more serious problem such as a brain tumor that requires surgical intervention. Painkillers may provide short-term relief, but the underlying problem will return unless an effort is made to treat its true source. Fault-based approaches to software quality treat the symptoms, but do not necessarily treat the

causes. We must look deeper to find and fix the errors, the cause of the problem, rather than finding and fixing only the fault, the symptom of the problem

Defect prevention techniques can be used during the creation of software artifacts to help developers create high-quality artifacts. These artifacts should have fewer faults that must be removed during inspection and testing. The Requirement Error Taxonomy that we have developed helps focus developers' attention on common errors that can occur during requirements engineering. By focusing on those errors, the developers will be less likely to commit them. This paper investigates the usefulness of the Requirement Error Taxonomy as a defect prevention technique. The goal was to determine if making requirements engineers' familiar with the Requirement Error Taxonomy would reduce the likelihood that they commit errors while developing a requirements document. We conducted an empirical study in which the participants were given the opportunity to learn how to use the Requirement Error Taxonomy by employing it during the inspection of a requirements document. Then, in teams, they developed their own requirements document. This requirements document was then evaluated by other students to identify any errors made. The hypothesis was that participants who find more errors during the inspection of a requirements document would make fewer errors when creating their own requirements document. The overall result supports this hypothesis.

2.1. Approaches on the Sources of Faults

Prior research has employed the sources of faults in different ways with varying levels of success. SE techniques use different methods to analyze faults to determine their causes and suggest process improvements [5, 12, 31, 54, 55, 57]. Here we describe three examples of such techniques.

Root Cause Analysis (RCA) helps identify systematic development problems to drive process improvement. These problems are detected during the testing phase and, separate from the development process, analyzed to determine their cause and suggest process improvements [12]. Four multi-dimensional *defect triggers* help developers characterize the source of the faults and identify process improvement needs based on those sources [51]. Our work extends this idea by emphasizing early-lifecycle faults (i.e. requirements) rather than late (i.e. implementation, testing).

Similarly, the *Orthogonal Defect Classification (ODC)* provides developers with in-process feedback on development activities. Developers classify defects using the ODC. Then, developers identify the *trigger* that caused the defect to surface (not necessarily the cause of defect insertion). Because this process is applied to code and the triggers explain the actions that revealed the failure, it is more objective than predicting the actual cause of defect insertion. The ODC has been shown to provide useful feedback to developers [23]. Our work extends this idea by helping developers understand not only what caused a fault to be revealed, but more importantly what caused the fault to be inserted.

A somewhat different approach to understanding the source of faults is *Error Abstraction*. Developers analyze faults detected during an inspection to determine their underlying cause, i.e. the error. Inspectors follow a process to abstract the faults to the errors that likely caused them. These errors then guide a re-inspection to detect other related faults that were overlooked during the original inspection. This process did not make use of a type of formal error taxonomy to guide inspectors [46]. Our work extends this approach by formalizing the error taxonomy, with input from psychology, to make developers more effective during the error inspection process.

In these methods, developers analyze only a sample of faults, potentially overlooking many errors. These methods also lack a formal process to assist developers in finding and fixing errors. A major drawback of many of these approaches is the lack of strong cognitive theory to describe the types of mistakes people make during development. Therefore, our work combines SE research with human error research to provide a more comprehensive solution [87, 95].

2.2. A Cognitive Psychology Perspective on Errors

Psychological study of human errors received increased attention beginning in the early 1970's (e.g., [80, 81]). Systematic models of human error capitalized on basic theoretical research in human cognition, especially related to information processing. It quickly became apparent that errors generally were not the result of irrational or maladaptive tendencies, but instead resulted from normal psychological processes gone awry. Reason [66] introduced a well-respected taxonomy of human errors that begins with a distinction between *slips*, *lapses*, and *mistakes.* Slips are often committed by experts who employ a familiar routine even though the situation calls for novel behavior. Mistakes are often committed by novices, who simply misunderstand something and act accordingly. To clarify this distinction, consider two programmers who are coding a *for-loop* in C. Arrays in C are referenced from zero and it is common to code a for-loop that walks through an array as for (var = 0; var < limit; var++). A novice programmer who does not understand array references in C might instead use for (var = 1; var <= limit; var++), a misunderstanding-based *mistake*. Conversely, an expert programmer might find himself in a situation where no initialization is needed (or desired) and should code for (var < limit; var++) but instead inserts the unwanted initialization anyway due to the activation of a familiar habit, an example of a *slip*.

Rasmussen [62, 64] used a related taxonomy to study errors in an organizational context. He distinguished between *skill-based errors, rule-based errors, and knowledge-based errors.* Rule-based errors arise when a familiar rule is applied inappropriately or an incorrect rule is employed. Knowledge-based errors occur when someone finds herself in an unfamiliar situation and must reason about how to behave. Gaps in their knowledge often lead to errors. This approach was employed by Bates and colleagues to identify the origins of medication errors and to provide alternative procedures in order to reduce adverse drug events [8].

Ko and Myers [42] used largely the same theoretical framework to adapt error models to software development tasks. Although their main focus was on programming errors made by novices, they identified a framework that encompassed both the specification and implementation stages. Likely because of their focus on programming errors by novices, the framework provides little specificity for errors that occur early in the lifecycle. Rather, it notes the possibility of a "skill-based error made during the creation of documents that result in a requirements specification error" and similar errors. True error taxonomies go beyond these abstract possibilities: "*The lead software engineer forgot to ask the client for input from end-users, so useful functionality was omitted from the requirements specifications*." Ko and Myers provide this rich analysis of errors only for novice programmers. Our target is different: to analyze errors made during the requirements engineering process (to prevent and catch costly mistakes early) and to work with more experienced developers who make fewer mistakes than novices.

3. REQUIREMENT ERROR TAXONOMY DEVELOPMENT AND EVELUATION

Lanubile et al., proposed the *Error Abstraction* approach, in which developers analyze faults detected during an inspection to determine the underlying errors likely to have caused them. These errors are then used to guide a re-inspection to detect additional faults. This work produced some promising initial results, but Lanubile, et al., did not pursue this research [12]. Our work built on Lanubile's approach by formalizing requirement error taxonomy, with additional input from cognitive psychology, to better support developers during the error abstraction and re-inspection process. Fig. 1 illustrates Dr. Walia's previous research in developing and evaluating the requirement error taxonomy. This work is discussed in following subsections.

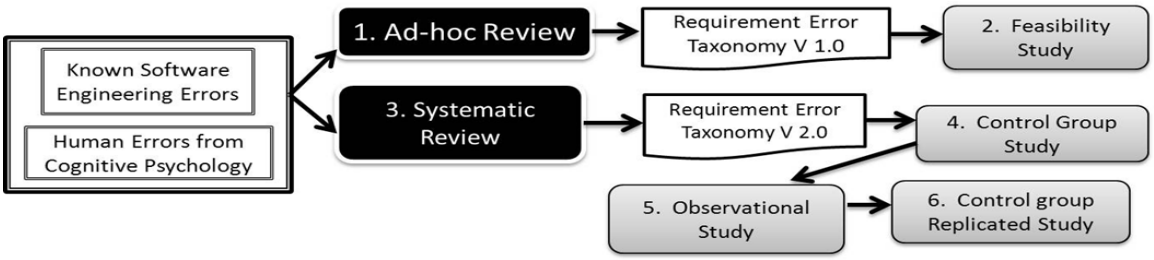

Figure 1. Research Approach

The requirement error taxonomy was developed by combining specific types of errors identified within the software engineering research together with cognitive psychology research about human errors. First, we developed an initial taxonomy (1. *Ad-hoc Review*) and empirically evaluated it (2. *Feasibility Study at Mississippi State University [MSU]*). Then, we refined the taxonomy using a more systematic approach (3. *Systematic Review*), and re-evaluated it using

three additional empirical studies (4. *Control Group Study* and 5. *Observational Study* at *MSU;* and 6. *Control Group Replicated Study at North Dakota State University [NDSU]*).

Because the systematic literature review and the Requirement Error Taxonomy (V2.0) have already been published (Walia, 2009), Section 3.1 just provides a summary of the taxonomy development. Section 3.2 provides an overview of the empirical studies and how the results motivated the current paper.

3.1. Development of Requirement Error Taxonomy

The requirement error taxonomy helps inspectors identify and understand errors and provides guidance for identifying additional errors and faults. It has evolved through two versions. We used an ad-hoc review of the software engineering and psychology literature to develop the initial requirement error taxonomy (V1.0) (Walia, 2006a). After the feasibility study (Walia, 2006b), we performed a *systematic literature review* to more completely identify and document the types of requirement errors (Walia, 2009). This review included 149 papers from software engineering, human cognition and psychology. The systematic approach is commonly used in other fields (e.g. medicine) to draw high-level conclusions across a series of detailed studies (Kitchenham, 2004).

To generate the taxonomy we grouped the errors identified in the software engineering, human cognition, and psychology literature into 14 detailed error classes which we then grouped into three high-level error types (Table 1). The three error types are *People Errors*, *Process Errors*, and *Documentation Errors*.

People Errors include mistakes caused by fallibilities of the individuals involved in project development. *Process Errors* are caused by mistakes in selecting the means of achieving goals/objectives and focus mostly on the inadequacy of the requirements engineering process.

Table 1 - Description of Requirement Error Classes (Walia, 2009)

Error Type	Error Class	Description
People Errors	Communication	Poor or missing communication among the various stakeholders
	Participation	Inadequate or missing participation of important stakeholders
	Domain Knowledge	Requirement authors lack knowledge or experience with problem domain
	Specific Application Knowledge	Requirement authors lack knowledge about specific aspects of the application
	Process Execution	Requirement authors make mistakes while executing requirement elicitation and development, regardless of the adequacy of the chosen process
	Other Cognition	Other errors resulting from the constraints on the cognitive abilities of the requirement authors
Process Errors	Inadequate Method of Achieving Goals and Objectives	Selecting inadequate or incorrect methods, techniques, or approaches to achieve a given goal or objective
	Management	Inadequate or poor management processes
	Elicitation	Inadequate requirements elicitation process
	Analysis	Inadequate requirements analysis process
	Traceability	Inadequate or incomplete requirements traceability
Documentation Errors	Organization	Problems while organizing requirements during documentation
	No Use of Standard	Problems resulting from the lack of using a documentation standard
	Specification	General documentation errors, regardless of whether the requirement author correctly understood the requirements

Documentation Errors are caused by mistakes in organizing and specifying the requirements regardless of whether the developer correctly understood the requirement.

Each error class was derived from specific errors identified in the software engineering and human cognition literature. Specifically, errors related to human cognition span all three error types and fall into seven of the fourteen error classes: *Domain Knowledge*, *Specific Application Knowledge*, *Process Execution*, *Inadequate Method of Achieving Objectives*, *Organization*, *Specification*, and *Other Cognition.*

The complete description of the systematic review process, details of the requirement error taxonomy and examples of errors and related faults for all 14 error classes can be found in the systematic literature review (Walia, 2009) and in Appendix A. To illustrate the error

taxonomy, we provide an example *participation error* along with a related fault taken from a generic ATM system:

Error: An important stakeholder (e.g., a bank manager in an ATM system) was not involved in the requirement gathering process;

Fault: Some functionality (e.g., handling multiple ATM cards simultaneously at different machines) was omitted.

3.2. Evaluation of Requirement Error Taxonomy as a Defect Detection Method

To evaluate the usefulness of the requirement error taxonomy and the error abstraction process, we conducted a family of four controlled experiments (studies one, two and three at MSU and study four at NDSU). A family of studies is a set of related studies that focus on the same research question or hypotheses (Basili, 1999). While each of our studies tested the same basic hypotheses, the designs of later studies were slightly modified based on the lessons learned during the earlier studies. As shown in Figure 2, the results from Study 1 motivated the design of Studies 2 and 3 (by adding a control group variable and an observational variable respectively). Similarly, the results from Study 2 motivated the design of Study 4 (by replicating Study 2 in a different university setting).

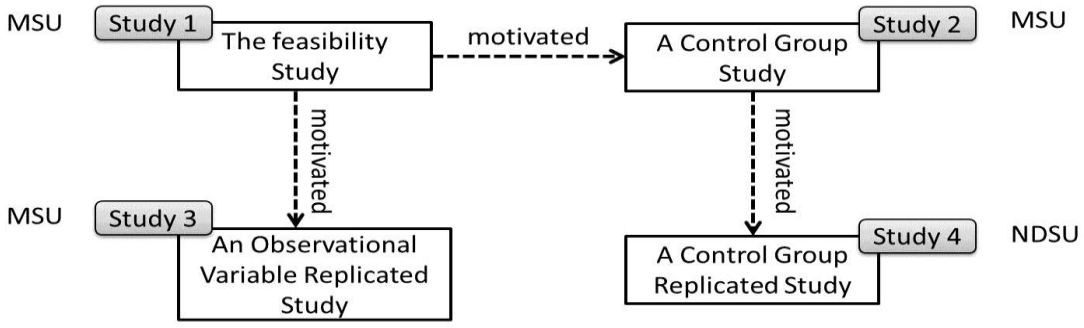

Figure 2. The Family of Experiments

We evaluated the usefulness of the requirement error taxonomy with four empirical studies. The validation goal of these studies was to ensure that: 1) the error classes are clearly described, useful, and complete, and 2) the developers can use the error taxonomy to increase their defect detection effectiveness during inspections.

Study 1 and 3 (see Fig. 1), were conducted in senior-level capstone courses where students developed a project for real customer. In these studies, the students first performed an inspection of their requirement document to identify faults. Then, they were trained on the use of error taxonomy. The students then used the error taxonomy to abstract and classify the errors that caused the observed faults. Finally, the students used the error information to guide the re-inspection of the requirement document. The results from these two studies indicated that the participants found the error taxonomy both easy to use and effective. In addition, by using the error taxonomy, the participants found a significant number of new faults during the re-inspection. Finally, most participants found errors that were derived from the cognitive psychology human error research, 10%-20% of the total errors reported [21, 26].

Study 2 and 4 (see Fig. 1), were conducted with students enrolled in graduate-level courses. In these studies, one group of students (i.e., the experiment group) used the same procedure as in the Study 1 and 3 described above. The other group of students (i.e., the control group) inspected the artifact two times without using error abstraction. In Study 2, the control group participants used the same fault inspection technique during both inspections and in Study 4, they used a more mature fault inspection technique for the re-inspection. We compared the results from the experimental group with the results from the control group to determine what portion of the additional faults found during the re-inspection can be attributed to the use of the error abstraction and classification approach. The results from these studies showed that the

group who used the error abstraction and classification process found significantly more faults during re-inspection than the control group, providing more evidence of its usefulness [25-26].

The results from these four studies can be summarized as follows: 1) the error abstraction and classification approach improves the effectiveness (number of faults found) of inspectors during a requirements inspection, 2) the requirement error taxonomy is subjectively useful for inspectors to find errors and faults, and 3) the human error research from cognitive psychology helped inspectors detect more faults. More details of the experiment designs and results from each of these studies can be referred [21-26].

While the requirement error taxonomy has been effective in detecting defects during inspections, a more useful analysis would require evaluating the effectiveness of the requirement error taxonomy for preventing defects from occurring during the requirements development. Leape, and other researchers have employed a similar approach to the analysis of adverse medical events in order to understand what caused the individuals to make errors [11, 13]. Leape et al., argued that the underlying cause of the problems should be used to prevent errors rather than attempting to remove the errors. Because the errors are mistakes or misunderstandings of the software engineers while creating a software artifact, the information about the commonly made errors can be used to educate software engineers to prevent them from making errors in the first place.

4. EXPERIMENT DESIGN

The major goal of this study is to evaluate the usefulness of the requirement error taxonomy as a defect prevention technique. This study investigates whether developers can avoid making errors if they have *a priori* information about the types of errors that can occur during requirement development.

This experiment is a control group design in which participants were divided into two groups and inspected a requirements document (developed externally) using the error abstraction and classification method (the first group) and using the traditional fault checklist method (the second group). These inspections resulted in a list of errors and faults for each participant. The participants are then briefed on the different types of errors that may occur during the requirement development (using information in the requirement error taxonomy). Next, each team of 3 or 4 participants developed a requirement document for a different system. These requirements documents were then evaluated by other participants to identify errors and faults. The details of the study are provided in the following subsections.

4.1. Research Questions and Hypotheses

Three research questions were investigated in this study.

Research Question 1: Does knowledge about the types of errors and faults that may occur during requirement development make developers less likely to make those errors?

Research Question 2: Does finding a particular type of error (e.g., people or process or documentation errors) reduce the likelihood that a developer will commit that type of error while creating his own document?

Research Question 3: Is a student's understanding of the error classes an accurate predictor of their effectiveness during an error-based inspection of someone else's document and an

accurate predictor of their ability to make fewer errors when developing their own document?

Three hypotheses related to above questions are:

Hypothesis 1: The teams whose members find more errors (as opposed to the faults) during the inspection of someone else's requirement document will make fewer errors and faults when creating their own document.

Hypothesis 2: The teams whose members find more errors of an error type during the inspection of someone else's document will make fewer errors of that error type when creating their own document.

Hypothesis 3: The teams whose members find significantly more People Errors during the inspection of someone else's document will make fewer total errors when creating their own document.

4.2. Independent and Dependent Variables

The experiment manipulated the following independent variables:

The pre-test: measures the performance of subjects during an in-class training exercise on the error taxonomy.

The inspection technique: employed by the participants prior to the development of the requirements document.

We also measured the following dependent Variables:

Effectiveness: the number of errors and the number of faults found by each subject.

4.3. Participating Subjects

Forty three undergraduate students enrolled in System Analysis and Design course at North Dakota State University (NDSU) participated in this study. The students worked in

thirteen teams of three (there was one team of four students) to develop a requirement document for different projects.

4.4. Artifacts

There were two phases to this study (inspection and development). Each phase used different artifacts. First, during the inspection phase, all participants inspected a software requirements specification (SRS) document developed externally by capstone project students at Florida International University (FIU). This requirements document describes an interactive restaurant menu (RIM) that the students at FIU later implemented. The RIM system provides customers the ability to make their dining choices through a table terminal, PDA, and/or web-based application. This document was chosen because it was developed by a team of four undergraduate students and contains a representative sample of the kind of errors and faults that the participants at NDSU would make. Second, for the development phase, each team of four participants developed a software requirement specification (SRS) document for a different system. Table 2 provides a list of these systems.

Table 2 – Teams and System Descriptions

Team #	System Description
1	Retail Management System
2	Healthcare Database System
3	Food Order Via Facebook
4	Online Playlist Management System
5	Incident Management System
6	Social Networking based Course Management System
7	Online Video Rental System
8	Mobile Payment and Account Management System
9	Scoreboard Content Management System
10	Concert Hall Management System
11	Portable Health Monitoring System
12	Warehouse Inventory Management System
13	NDSU Biometric System
14	Bison Campus Maintenance Alert System

4.5. Experiment Methodology

The experimental design includes five steps. Fig. 3 provides an overview of experiment steps. Fig. 4 shows the details of the experiment steps. The details are provided in the following subsections.

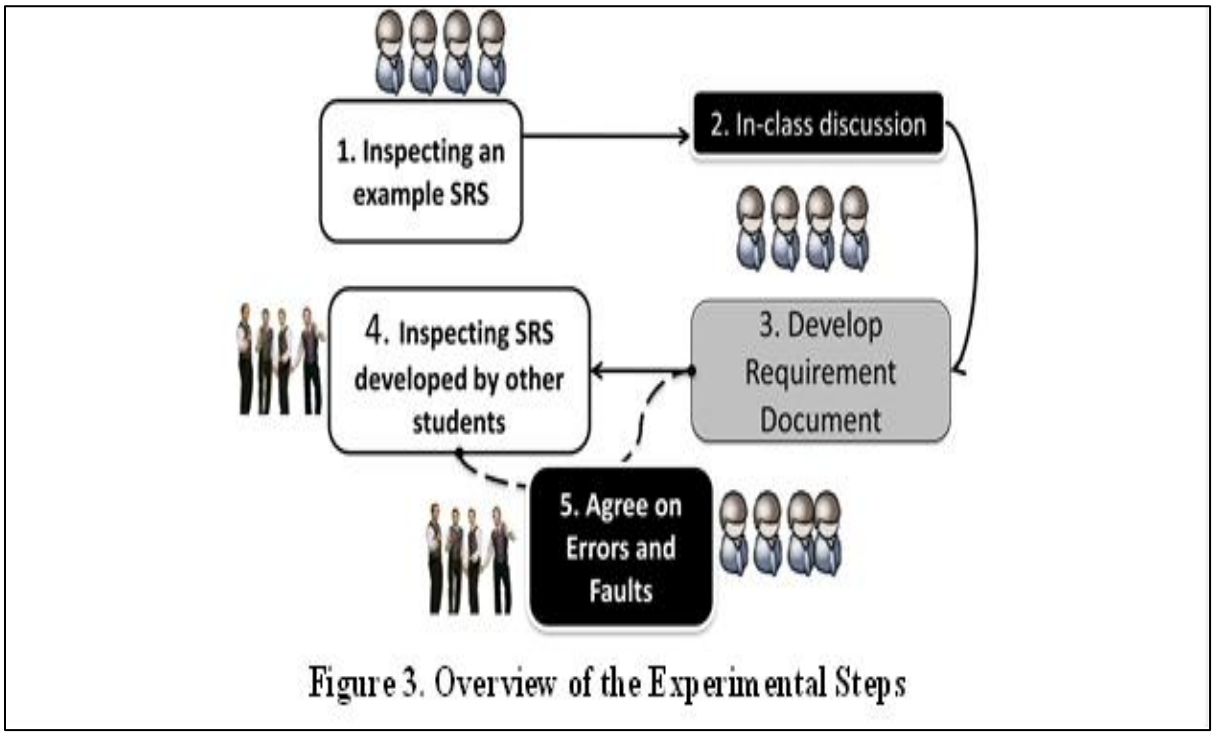

Figure 3. Overview of the Experimental Steps

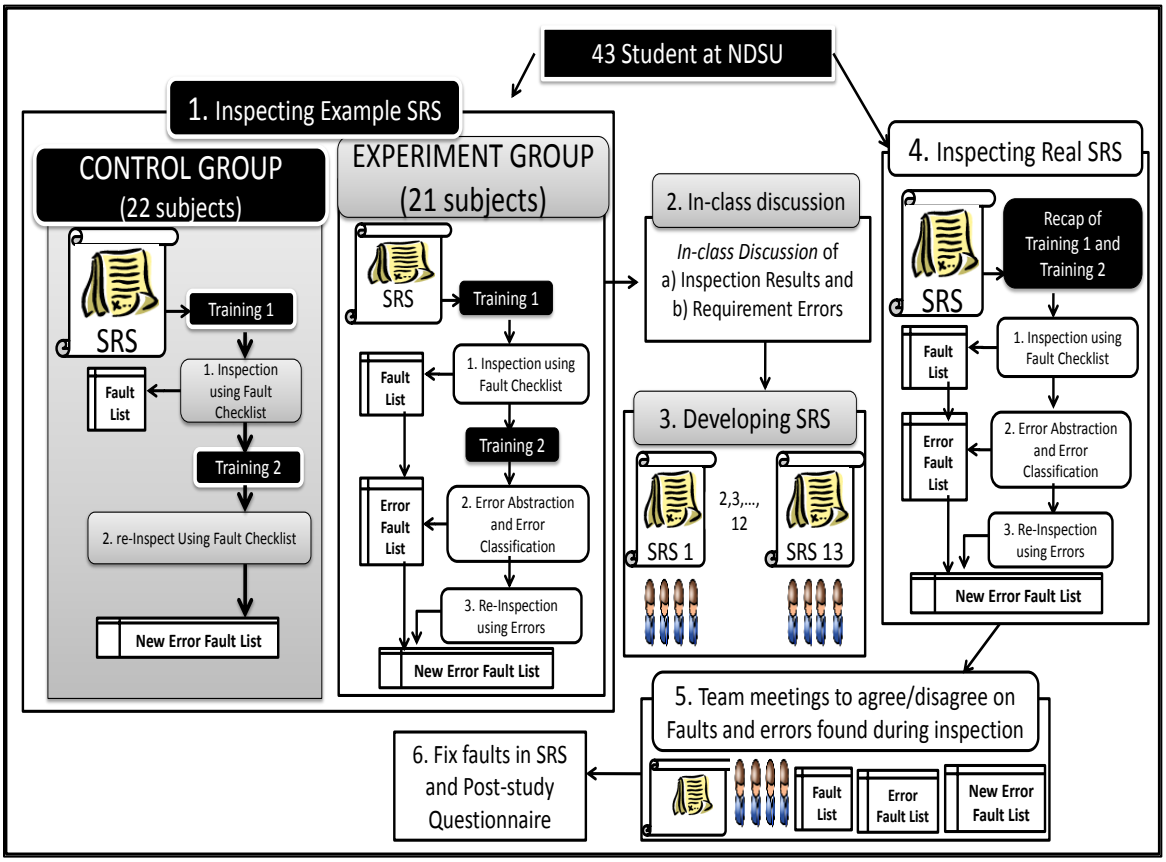

Figure 4. Details of the Experimental Procedure

4.5.1. Step 1- Inspecting an Example SRS Document

This step involved using the error abstraction and classification method to inspect the RIM SRS document:

4.5.1.1. *Training 1:* During this 60 minutes session, the participants received the SRS document for the RIM system, the fault checklist and a list of the fault classes. They were instructed on how to use the fault checklist to locate faults and how to record faults.

4.5.1.2. *Inspecting SRS for Faults*: Using the information from Training 1, each participant inspected the RIM requirement document using the fault checklist. This step produced 43 individual fault lists (one per participant).

Training 2: During this training session, the participants were divided into two groups. The control group had 22 subjects and the experiment group had 21 subjects. The 22 subjects in the control group were re-trained on the fault checklist technique and were told that they had missed certain faults during the first inspection. The 21 participants in the experiment group learned about the error abstraction process and the requirement error taxonomy. The participants were first trained on how to use the error abstraction process to abstract errors from the faults on their individual fault lists. Because abstracting errors from faults is a subjective process, the requirement error taxonomy was described in detail to support the error abstraction process. Next, the participants were taught how to classify errors. They were given the description of an example system and 12 errors. They classified those errors using the 14 detailed error classes in the taxonomy. The participants' classifications of these errors served as a pre-test to provide an idea of how well they understood the classification scheme. They were also instructed on how to use the error list (to record and classify the abstracted errors). Finally, the participants were taught how to use the error information in the error list to re-inspect the requirements document. They were also instructed on how to record the new faults found during re-inspection in the new error-fault list.

4.5.1.3. *Re-inspection of SRS Document:* The 21 participants used the information about errors gathered during the *error abstraction and classification* to re-

inspect the requirements document to locate additional faults. The output of this step was 21 individual new error-fault lists (one per experiment group participant). Also, the 22 participants in the control group re-inspected the SRS document using the same fault checklist technique and produced a list of new faults found during the second inspection. The output of this step was 22 individual new fault lists (one per control group participant)

4.5.2. Step 2- In class Discussion of Inspection Results

Using the information about the errors and faults found in Step1, the first author discussed these errors and faults separately with the participants in the experiment and the control group.

4.5.3. Step 3- Development of SRS Documents

The 43 participants were divided into 14 teams (7 belonging to the control group and 7 belonging to the experiment group). The 21 experiment group subjects were divided into 7 teams of three subjects each. The 22 control group subjects were also divided into 7 teams where 6 teams were made of three subjects and 1 team with four members. The participants were allowed to select their own team members. Each team developed the requirements document for a different software system (as described in Table II). Team numbers 1, 3, 5, 7, 9, 11, 13 belonged to the experiment group and Team numbers 2, 4, 6, 8, 10, 12, 14 belonged to the control group.

4.5.4. Step 4- Inspection of Developed SRS Documents

During this step, the instructor provided the participants with a training session on the Fault Checklist, Error Abstraction Process and Requirement Error taxonomy. After each team had developed their software requirement document, three or four participants (equivalent to the number of developers) were assigned to inspect it. These inspectors

were chosen at random with the constraint that they had to all come from different development teams (and that they should not be inspecting the document that they were part of the development). Each participant then used the error abstraction and classification method to inspect the requirements document (in the same manner as in Step 1). Each participant produced three outputs: 1) individual fault list (first inspection), 2) individual error-fault list (error abstraction), and 3) individual new fault list (re-inspection). This step resulted in a list of errors and faults found by four different inspectors for each of the thirteen requirement documents (43 in total).

4.5.5. Step 5- Team Meetings to Discuss Errors and Faults

The developers of each requirements document analyzed and discussed the errors and faults found by the inspectors. This step resulted in a list of errors and faults that the development team agreed with and a list they disagreed with, along with the reason for disagreement. The first author discussed this list with the developers and the inspectors (if the description of error/fault was unclear) to arrive on a final list of agreed-upon errors and faults.

4.5.6. Step 6- Fix SRS and Post-Study Questionnaire

This step used the list of agreed error and faults from Step 5 to fix the problems in their documents. The subjects were then given a questionnaire to provide feedback about the error abstraction process, the requirement error taxonomy, and the quality of SRS documents.

4.6. Data Collection

This section provides a brief description of qualitative and quantitative data collected during Study. The quantitative data includes the faults found by participants prior to the

development of their own document. This includes the faults found during Step 1 using the fault checklist technique (i.e., the control group subjects) and the faults and errors found by the participants using the error abstraction and classification process (i.e., the experiment group subjects). We then collected the faults made by the student teams during the development of their SRS documents (Table II). Each of these 14 documents was inspected by students and their fault lists were analyzed to count the number of unique faults committed during the development of these documents.

The fault reporting forms also provide the participants with space to indicate timing information, including the start and end times of the inspection, the time they found each fault, and any breaks they took.

In addition, the fault reporting forms required the participants to rate the *importance level* and the *severity* of the faults identified during the first and second inspection cycle. The importance level, which is the potential that a particular requirement fault found during inspection can cause a redesign of the software, was classified using the following scale (0- not important, designer should easily see the problem; 1- problem, if a failure occurs it should be easy to find and fix; 2– important, if a failure occurs it could be hard to find and fix, 3– very important, if a failure occurs it could be very hard to find and fix, 4- if a failure occurs it could cause a redesign). The severity, which is the probability that a particular fault will cause a system failure, was classified using the following scale (0- will not cause failure, regardless whether it is caught by the designer; 1- will not cause failure, because it will be caught by the designer; 2– could cause a failure, but most likely be caught by the designer, 3– would cause a failure, will most likely not be caught by the designer).

One of the researchers validated that the faults reported by each participant were true-positives. The researcher, who had knowledge of the system for which the requirements were

developed, read through the faults reported by each participant to remove any false-positives before analyzing the data. If any faults were unclear, the researcher clarified them with the participant to accurately determine the validity of the fault. Regarding the evaluation of the errors reported by participants, the researcher read through the errors to ensure that the description of each error represented a real mistake or misunderstanding that could have happened during the development. Also, the researcher evaluated the errors for correct classification by making sure that the description of the error was consistent with the actual description of that error class.

5. RESULTS AND ANALYSIS

This section provides an analysis of error data. This section is organized to test the effect the inspection in Step 1 had on the subsequent steps. We also compared whether information about errors (the experiment group) vs. faults (the control group) prior to the development are significantly correlated to the number of errors and faults present in the developed documents. An alpha value of 0.05 was used for all statistical tests.

5.1. Analysis of the Inspection Prior to the Development vs. Quality of Requirements Document Developed by Student Teams

This section analyzes the effect the number of errors found by developers (belonging to the experiment group) during an error-based inspection of someone else's requirement document (i.e., Step 1) had, on the number of errors and faults committed by them while developing their own requirements documents (i.e., Step 3). Similarly, we also analyzed the number of faults found by developers during (belonging to the control group) a fault-checklist based inspection of someone else's requirement document (i.e., Step 1) had, on the number of errors and faults committed by them while developing their own requirements documents (i.e., Step 3). The goal of this analysis was to evaluate whether developers with prior knowledge of errors vs. fault had a correlation (we expected a negative correlation) with the number of defects present in their own documents.

Because each development team consisted of four people and the inspection data from Step 1 was individual data, we had to combine the Step 1 scores into one team score. The error-detection effectiveness of the development teams (the experiment group) and the fault detection effectiveness of the teams (the control group) during Step 1 was calculated by combining the list

of unique errors and/or unique faults each participant found in the *RIM* requirement document. The errors committed by developers while developing their own requirements document (during Step 3) were calculated by combining the list of unique errors that the four inspectors found (during Step 4) and agreed upon by the developers and the inspectors (Step 5). This analysis was performed for each of the 14 teams. We performed this analysis separately for the 7 teams in the experiment group (where the error and fault count during step 1 were analyzed against the errors and faults during the step 3) and for the 7 teams in the control group (where the fault count during step 1 were correlated against the errors and faults during Step 3).

The reason for using the unique errors and faults (as opposed to the total error and fault count) are because we were interested in coverage of the error space by the teams (and the fault space by the control group teams) as opposed to high overlap in the error and fault lists or individual team member's knowledge.

Figure 5 plots the number of unique errors found during an inspection *prior to the development* against the number of unique errors and the number of faults made *during the*

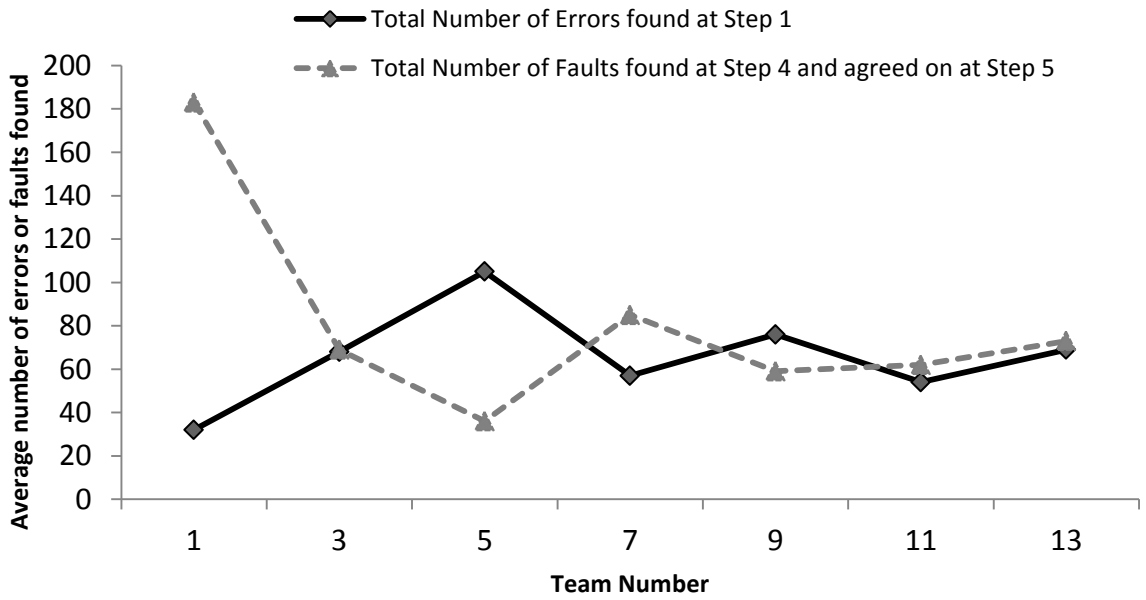

Figure 5. Error Detection Pre-Development vs. Faults during the Development

development of their requirement document for the 7 teams in the experiment group.

To test the hypothesis 1, we ran a linear regression test to see whether the number of errors found by a team during Step 1 is inversely correlated to the number of errors made during development of their own requirements. The results from a linear regression shows a strong and significant correlation ($r = -0.831$, $r^2 = 0.69$ and p-value of 0.013) between the number of errors found by the development team and the number of faults made by them during the development. That is, the teams that found a larger number of errors prior to the development made fewer errors and faults during the development of their own document and vice versa.

Similarly, Figure 6 plots the number of unique faults found during the inspection *prior to the development* against the number of unique errors and the number of faults made *during the development* of their requirement document for the 7 teams in the control group.

We ran a linear regression test to see whether the number of faults found by a team during Step 1 is inversely correlated to the number of errors made during development of their own requirements. The results from a linear regression shows a weak but positive correlation (r =

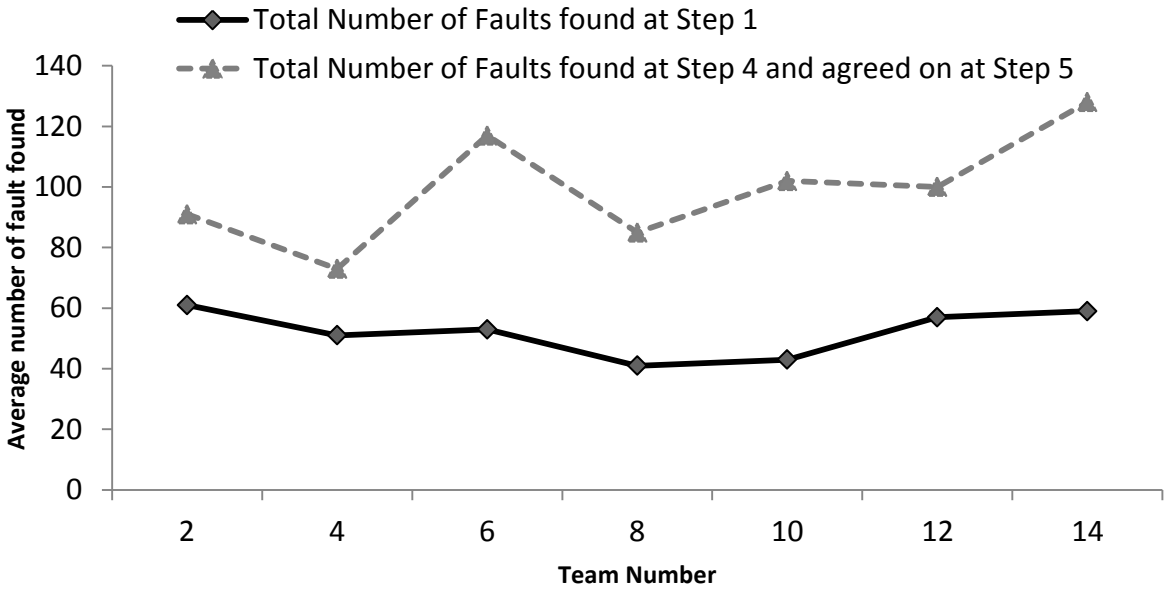

Figure 6. Fault Detection Pre-Development vs. Faults during the Development

+0.352, r^2 = 0.124 and p-value of 0.13) between the number of faults found by the development team and the number of faults made by them during the development. That is, the teams that found a fewer (or larger) number of errors prior to the development made fewer (or more) faults during the development of their own document. This result was not significantly correlated (p-value of 0.2192).

Based on these results, using the requirement error taxonomy to guide an inspection of someone else's document does appear to reduce the number of errors (and resulting faults) committed during the subsequent development of a different document. However, the fault checklist technique alone cannot help the developers avoid making same faults when developing their own document. A major reason for this might have been the fact that, using the error inspection, developers are able to understand the root cause of the faults (and the mistakes that are likely to happen during the development) as opposed to the manifestation of the errors during the fault based inspection.

5.2. Analysis of the Type of Errors Detected Prior to the Development of the Requirements Document

While Figure 5 shows the total number of errors, it is also important to conduct a similar analysis for each major error type. We analyzed whether the number of errors within each error type (*People* or *Process* or *Documentation Errors*) detected by developers prior to development had an effect on the faults committed during the development.

We broke down the total errors into the number of *People*, *Process*, and *Documentation Errors* found by each team prior to development and the number of *faults caused by each error type* made during development.

To test hypothesis 2, we ran a linear regression test to see whether the number of errors belonging to each error type found at Step 1 is inversely correlated to the number of faults (caused by the errors of that error type) made during the development. The results show that only the *People Errors* found at Step 1 are significantly correlated to the number of faults made by teams that were attributable to the *People Errors* while developing their own documents. The result had a significant negative correlation ($r = -0.743$, $r_2 = 0.56$, and p-value = 0.03). While the *Process Errors* and *Documentation Errors* were negatively correlated, the results showed a weak and an insignificant correlation.

Based on this result, the teams that found a larger number of *People Errors* prior to the development made fewer numbers of faults when developing their own document. Therefore, *People Errors* are the most common type of errors and a major source of requirement problems. Therefore, during requirement creation, developers can use the error taxonomy to focus their attention on commonly occurring *people errors*, so that they will be less likely to commit them, resulting in high-quality software artifacts.

5.3. Analysis of the Pre-Test Data

The number of errors correctly classified during the pre-test at Step 1 was analyzed to determine whether it was related to the number of errors found during Step 1. Note that only the 21 subjects in the experiment group performed the error inspection during the Step 1. The goal of this analysis was to understand whether performance of a team on a pre-test could be an accurate predictor of their performance during the actual development. To perform this analysis, the average number of errors correctly classified by four participants on each team was compared against the number of errors committed during development. The linear regression test showed

that the two variables had a positive relationship but the correlation was weak and was not significant.

6. THREATS TO VALIDITY OF RESULTS

In this study, we were able to address some threats to validity. To avoid a learning effect during the pre-test, the order of the errors being classified was randomized. Also, the participants inspected the requirements for a real system prior to the development as opposed to using an artificially seeded document. While the participants developed real documents, there were some threats to external validity. The study focused on students in a classroom setting who are likely to have different experience and time pressures than would be true of professionals in a real environment. Also, there remains a threat to external validity of using a requirements document that was not implemented. This study was an initial investigation and we plan to address this threat in future study. We were able to address the threat by including a control group. Finally, we do not know the actual number of errors and faults present in the documents developed by the participants. We only used the number of errors and faults found by the inspectors. So, there might be more errors present in the document that could change the results.

7. DISCUSSION OF RESULTS

A major focus of this study is to investigate the usefulness of the error taxonomy (as opposed to the fault checklist technique) as a defect prevention technique.

Our first question focused on understanding whether the participants made fewer errors and faults in their own documents after they had found errors (vs. faults) in the earlier inspection of someone else's document. The results in Figure 5 and Figure 6 showed that the performance (of participants in each team) during an error based inspection prior to the development had an inverse relationship with the number of errors and faults made during the development of their own documents. This result means that the more errors found during an inspection (Step 1) the fewer errors and faults made during subsequent development of a requirements document. When this result was tested statistically there was a significant and strong negative correlation between the two variables. There was no significant and negative correlation between the performances of student teams in control group. *Therefore, using error taxonomy to guide an inspection prior to development of a requirement document is beneficial.*

The results from our previous studies have shown that even though the three error types in taxonomy provide a good coverage of requirement error space, People Errors tend to be the largest source of requirement problems [22-26]. Therefore, this research question investigated the effect of the number of People, Process, and Documentation Errors found prior to the development, on the number of errors committed within each error type, and the total number of faults committed (within each error type) while developing their own documents. The answer to this question will help software developers focus their attention on the most common errors during the requirement development process. The results show that among the three errors types, *People Errors* had the largest and most significant effect. The teams that found significantly

larger number of *People Errors* during an inspection of someone else's document, made fewer *People Errors* and fewer total number of errors while developing their own documents.

Software organizations can use the error taxonomy to educate their software developers about the common errors that can occur during the artifact creation process. By focusing on these errors, developers will be less likely to commit them. Furthermore, because the error taxonomy describes some common faults that can result from the errors, the artifact creators can use this information to reduce the chance they will insert those faults into the artifact. As a result, they will produce higher quality documents that will require less effort to remove the smaller number of faults during the inspection and testing phases. Especially in large software organizations, major problems arise from the mistakes and misunderstandings among the people involved in the development process. Furthermore, understanding the commonly occurring errors in an organization over a period of time can help correct the system inadequacies that cause the individuals to make errors

8. CONCLUSION AND FUTURE WORK

Based on the results provided in this paper, investing in the error taxonomy to help developers learn from other's mistakes is a reasonable cost for avoiding costly rework, that is fixing problems that should have been fixed in earlier lifecycle phases or should have been prevented altogether. The results in this paper have motivated us to further investigate the promise of using the error taxonomy as a defect prevention technique. In future, we plan to replicate this study and other studies to investigate the use of error taxonomy as defect prevention technique in different settings including capstone project courses (where students will actually implement the systems) and in an industrial setting (with professional software developers). Our future work also includes creating more formal techniques for error prevention using the requirement error taxonomy.

REFERENCES

1. IEEE Std 610.12-1990, IEEE standard glossary of software engineering terminology. 1990.
2. Basili, V.R., Green, S., Laitenberger, O., Lanubile, F., Shull, F., Sørumgård, S., and Zelkowitz, M.V., "The Empirical Investigation of Perspective-Based Reading." *Empirical Software Engineering: An International Journal*, 1996. 1(2): 133-164.
3. Boehm, B. and Basili, V.R., "Software Defect Reduction Top 10 List." *IEEE Computer*, 2001. 34(1): 135-137.
4. Card, D.N., "Learning from our mistakes with defect causal analysis." *Software, IEEE*, 1998. 15(1): 56-63.
5. Chillarege, R., Bhandari, I.S., Chaar, J.K., Halliday, M.J., Moebus, D.S., Ray, B.K., and Wong, M.Y., "Orthogonal defect classification-a concept for in-process measurements." *IEEE Transactions on Software Engineering*, 1992. 18(11): 943-956.
6. Florac, W. *Software Quality Measurement: A Framework for Counting Problems and Defects*. Technical Reports, CMU/SEI-92-TR-22. Software Engineering Institute: 1992.
7. Grady, R.B., "Software Failure Analysis for High-Return Process Improvement," *Hewlett-Packard Journal*, 1996. 47(4):15-24.
8. Jacobs, J., Moll, J.V., Krause, P., Kusters, R., Trienekens, J., and Brombacher, A., "Exploring Defect Causes in Products Developed by Virtual Teams." *Journal of Information and Software Technology*, 2005. 47(6): 399-410.
9. Kan, S.H., Basili, V.R., and Shapiro, L.N., "Software Quality: An Overview from The Perspective Of Total Quality Management." *IBM Systems Journal*, 1994. 33(1): 4-19.
10. Kitchenham, B. *Procedures for Performing Systematic Reviews*. TR/SE-0401. Department of Computer Science, Keele University and National ICT, Australia Ltd.: 2004.

11. Kohn, L.T., Corrigan, J.M., and Donaldson, M.S., "To Err is Human: Building a Safer Health System. A Report of the Committee on Quality Health Care". 2000, Washington, DC.
12. Lanubile, F., Shull, F., and Basili, V.R. "Experimenting with error abstraction in requirements documents". In *Proceedings of Fifth International Software Metrics Symposium, METRICS98*. p. 114-121.
13. Leape, L. L., "Errors in Medicine," *Journal of the American Medical Association,* 272(23): 1851-1857. 1994
14. Lezak, M., Perry, D., and Stoll, D. "A Case Study in Root Cause Defect Analysis". In *Proceedings of the 22nd International Conference on Software Engineering*. Ireland. 2000. p. 428-437.
15. Masuck, C., "Incorporating A Fault Categorization and Analysis Process in the Software Build Cycle*." Journal of Computing Sciences in Colleges* 2005. 20(5): 239 – 248.
16. Mays, R.G., Jones, C.L., Holloway, G.J., and Studinski, D.P., "Experiences with Defect Prevention*." IBM Systems Journal*, 1990. 29(1): 4 – 32.
17. Nakashima, T., Oyama, M., Hisada, H., and Ishii, N., "Analysis of Software Bug Causes and Its Prevention*." Journal of Information and Software Technology*, 1999. 41(15): 1059-1068.
18. Norman, D.A., "Categorization of Action Slips*." Psychological Review*, 1981. 88: 1-15.
19. Rasmussen, J., "Skills, Rules, Knowledge: Signals, Signs and Symbols and Other Distinctions in Human Performance Models*." IEEE Transactions: Systems, Man, & Cybernetics*, 1983. 257-267.
20. Reason, J., *Human Error*. 1990, New York: Cambridge Press.
21. Sakthivel, S., "A Survey of Requirements Verification Techniques," *Journal of Information Technology*, 668-79. 1991

22. Walia, G.S., Empirical Validaton of Requirement Error Abstraction and Classification: A Multidisciplinary Approach, M.S Thesis, Computer Science and Engineering, Mississippi, Starkville, 2006(a).
23. Walia, G.S., Carver, J., and Philip, T. "Requirement Error Abstraction and Classification: An Empirical Study". In *Proceedings of IEEE Symposium on Empirical Software Engineering*. Brazil: ACM Press. 2006(b). p. 336-345.
24. Walia, G.S. and Carver, J., "A Systematic Literature Review to Identify and Classify Requirement Errors," *Journal of Information and Software Technology*, 2009. 51(7) 1087-1109.
25. Walia, G., Carver, J., and Philip, T., Requirement Error Abstraction and Classification: A Control Group Replicated Study, in *18th IEEE Symposium on Software Reliability Engineering*. 2007: Sweden.
26. Walia, G., Carver, J., Using Error Abstraction and Classification to Improve the Quality of Requirements: Conclusions from Family of Studies, Technical Report. 2010, NDSU, http://cs.ndsu.edu/research/ reports.htm

APPENDIX

This appendix describes the different errors in each of the fourteen detailed error classes (described in Table 1). A complete description of the requirement error taxonomy (along with examples of errors and faults) has been published in a systematic literature review (Walia, 2009). The list below shows each error class along with the specific errors that make up that error class.

Communication Errors

- Inadequate project communications
- Changes in requirements not communicated
- Communication problems, lack of communication among developers and between developers and users
- Poor communication between users and developers, and between members of the development team
- Lack of communication between sub teams
- Communication between development teams
- Lack of user communication
- Unclear lines of communication and authority
- Poor communication among developers involved in the development process
- Communication problems, information not passed between individuals
- Communication errors within a team or between teams
- Lack of communication of changes made to the requirements
- Lack of communication among groups of people working together

Participation Errors

- No involvement of all the stakeholders
- Lack of involvement of users at all times during requirement development
- Involving only selected users to define requirements due to the internal factors like rivalry among developers or lack of the motivation
- Lack of mechanism to involve all the users and developers together to resolve the conflicting requirements needs

Domain Knowledge Errors

- Lack of domain knowledge or lack of system knowledge
- Complexity of the problem domain

- Lack of appropriate knowledge about the application
- Complexity of the task leading to misunderstandings
- Lack of adequate training or experience of the requirement engineer
- Lack of knowledge, skills, or experience to perform a task
- Some properties of the problem space are not fully investigated
- Mistaken assumptions about the problem space

Specific Application Errors

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

Process Execution Errors

- Mistakes in executing the action sequence or the requirement engineering process, regardless of its adequacy
- Execution or storage errors, out of order sequence of steps and slips/lapses on the part of people executing the process

Other Human Cognition Errors

- Mistakes caused by adverse mental states, loss of situation awareness
- Mistakes caused by ergonomics or environmental conditions
- Constraints on humans as information processors e.g., task saturation

Inadequate Method of Achieving Goals and Objectives

- Incomplete knowledge leading to poor plan on achieving goals
- Mistakes in setting goals
- Error in choosing the wrong method or wrong action to achieve goals
- Some system-specific information was misunderstood leading to the selection of wrong method
- Selection of a method that was successful on other projects
- Inadequate setting of goals and objectives
- Error in selecting a choice of a solution

- Using an analogy to derive a sequence of actions from other similar situations resulting in the wrong choice of a sequence of actions
- Transcription error, the developer understood everything but simply made a mistake

Management Errors

- Poor management of people and resources
- Lack of management leadership and necessary motivation
- Problems in assignment of resources to different tasks

Requirement Organization Errors

- Poor organization of requirements.
- Lapses in organizing requirements.
- Ineffective method for organizing together the requirements documented by different developers.

Requirement Traceability Errors

- Inadequate/poor requirement traceability
- Inadequate change management, including impact analysis of changing requirements

Requirement Elicitation Errors

- Inadequate requirement gathering process
- Only relying on selected users to accurately define all the requirements
- Lack of awareness of all the sources of requirements
- Lack of proper methods for collecting requirements

Requirement Analysis Errors

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

No Use of Standard for Documenting Errors

- No use of standard format for documenting requirements
- Different technical standard or notations used by sub teams for documenting requirements

Specification Errors

- Missing checks (item exists but forgotten)
- Carelessness while organizing or documenting requirement regardless of the effectiveness of the method used
- Human nature (mistakes or omissions) while documenting requirements
- Omission of necessary verification checks or repetition of verification checks during the specification