Evaluating the Use of Requirement Error Abstraction and Classification Method for Preventing Errors during Artifact Creation: A Feasibility Study

Gursimran S. Walia
Computer Science Department
North Dakota State University
Fargo, USA
gursimran.walia@ndsu.edu

Jeffrey C. Carver
Department of Computer Science
University of Alabama
Tuscaloosa, USA
carver@cs.ua.edu

Abstract— **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. Our claim is that, 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 of four, 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.**

Keywords-software errors, error abstraction, requirement error taxonomy, error prevention, empirical study.

I. 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 a 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 and 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.

II. Background on Error Abstraction and Requirement Error Taxonomy

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].

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 a requirement error taxonomy, with additional input from cognitive psychology, to better support developers during the error abstraction and re-inspection process. Fig. 1 illustrates our previous research in developing and evaluating the requirement error taxonomy. This work is briefly discussed in Section II.A.

A. Development of Requirement Error Taxonomy

The requirement error taxonomy has evolved through two versions. To create the initial version (V1.0), we performed an ad-hoc review of the software engineering and psychology literature to identify and classify requirement errors [22]. Next, this taxonomy was empirically evaluated to determine its usefulness to support the error abstraction and

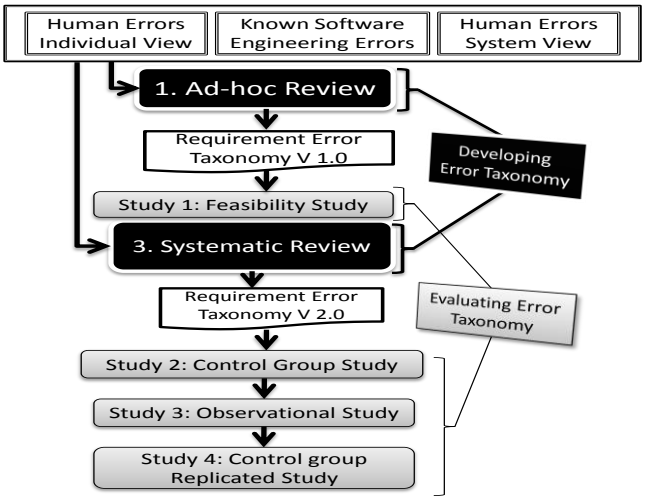

Figure 1. Process of Developing and Evaluating Requirement Error Taxonomy

re-inspection process [23]. After establishing the feasibility of such an approach, a more formalized, systematic literature search of software engineering and cognitive psychology research was performed to refine the error taxonomy. The systematic review, commonly used in medicine, is a process for documenting high-level conclusions that can be derived from a series of detailed studies [10]. The systematic review identified 149 papers (from software engineering, human cognition, and psychology) that provided insights into evolving the requirement error taxonomy into V2.0 [24].

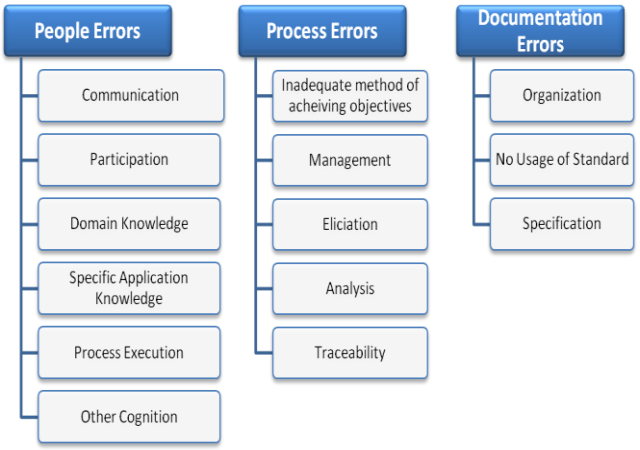

Figure 2. Requirement Error Taxonomy V 2.0 [24]

The errors identified from software engineering and cognitive psychology research were analyzed for similarities, and grouped into fourteen error classes (as shown in Fig. 2). These error classes were then classified into three high-level error types: *People Errors* (arise from the fallibilities of the people involved in the development process), *Process Errors* (arise when selecting the appropriate processes for achieving the desired goals, relate mostly to the inadequacy of the requirement engineering process), and *Documentation Errors* (arise from mistakes in organizing and specifying the requirements, regardless of whether the developer properly understood the requirements).

To illustrate the information contained in the error taxonomy, an example **participation error** (one of the People Errors) along with related faults is described here:

Error: An important stakeholder (e.g., a bank manger 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.

The complete systematic review process, the organization of the requirement errors into the error taxonomy, and the details of the requirement errors (along with examples of errors and faults) can be found in the systematic review publication [24].

B. *Evaluating the Requirement Error Taxonomy for Detecting Defects during Inspections*

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.

III. 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 repeated-measure design in which participants inspect a requirements document (developed externally) using the error abstraction and classification method. 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 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.

A. *Experiment Methodology*

1) *Research Questions and Hypotheses*

Three research questions were investigated in this study.

Research Question 1: Does knowledge about the types of errors 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 during the inspection of someone else's requirement document will make fewer errors 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.*

2) Variables

The experiment manipulated the independent variable.

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

We also measured the following dependent Variables:

Effectivenss*:* the number of errors and the number offaults found by each subject.

3) Participating Subjects

Fifty two 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 four to develop a requirement document for different projects.

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 I provides a list of these systems.

TABLE I. TEAMS AND SYSTEM DESCRIPTION

Team #	System Description
1	Selling and Tracking Ticket Sales for a Stadium Event
2	Gym Registration and Records System
3	Video Rental System
4	Car Rental System
5	Package shipping and tracking system
6	Manufacturing Package Contents Information Verification
7	Vending Machine Supply System
8	Online Banking System
9	Flight Management System
10	Online Test Taking System
11	Record Keeping System for Business
12	Online Shopping for Digital Downloads
13	Personnel Management System

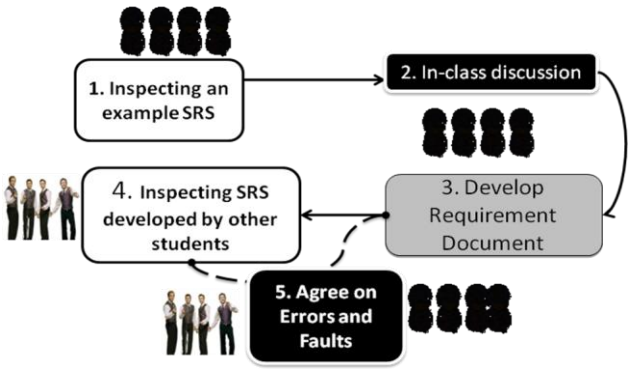

Figure 3. Overview of the Experimental Steps

B. Experimental Procedure

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.

1) Step 1- Inspecting An Example SRS Document:

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

a) 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.

b) Inspectiong SRS for Faults: Using the information from Training 1, each participant inspected the *RIM* requirement document using the fault checklist. This step produced 52 individual fault lists (one per participant).

c) Training 2: During this 90 minutes training session, participants 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 thieir individual fault lists. Because abstracting errors from faults is a subjectve 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 reinspect the requirements document. They were also instructed on how to record the new faults found during re-inspection in the *new error-fault list*.

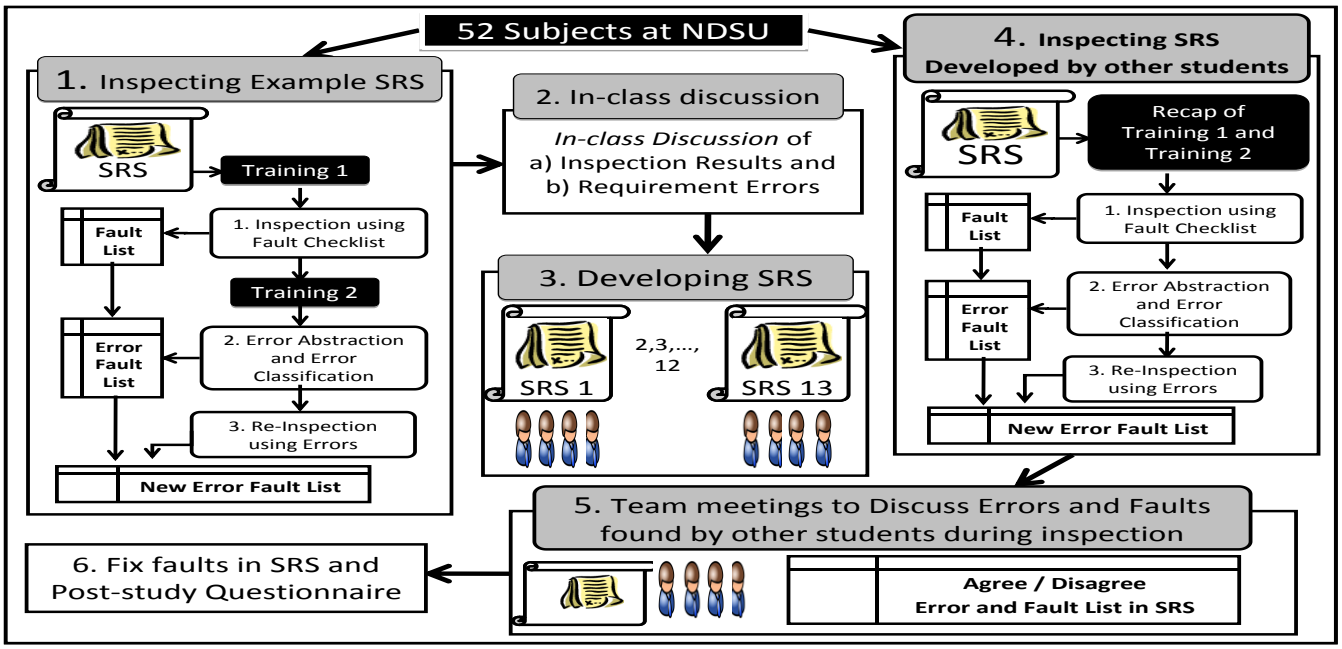

Figure 4. Details of the Experimental Procedure

d) Error Abstraction and Classification: The participants used the knowledge from Training 2 to abstract errors from the faults on their individual fault lists. The output of this step was 52 individual error-fault lists (one per participant).

e) Re-inspection of SRS Document: The participants used the information about errors gathered during the *error abstraction and classification* to reinspect the requirements document to locate additional faults. The output of this step was 52 individual new error-fault lists (one per participant).

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 with the participants. He also used the requirement error taxonomy to brief them on additional errors and faults that may occur during requirement development.

3) Step 3- Development of SRS Documents

The 52 participants were divided into 13 four-person teams. 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 I).

4) Step 4- Inspection of Developed SRS Documents

During this step, the first author provided the participants with a short training session to recap the *Fault Checklist*, *Error Abstraction Process* and *Requirement Error taxonomy*. After each team had developed their software requirement document, four participants were assigned to inspect it. The four inspectors were chosen at random with the constraint that they had to all come from different development teams. 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 (52 in total).

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 four 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.

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.

IV. Analysis and Results

This section provides an analysis of error data. This section is organized to test the effect that the errors found during the inspection in Step 1 had on the subsequent steps. An alpha value of 0.05 was used for all statistical tests.

A. Analysis of the Effect of the Number of Errors Detected Prior to Developing Requirement Document

This section analyzes the effect the number of errors found by developers 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).

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 team during Step 1 was calculated by combining the list of unique errors 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 13 teams. The reason for using the unique errors (as opposed to the total error count) is because we were interested in coverage of the error space by the team as opposed to high overlap in the error 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 development* of their requirement document. The results are

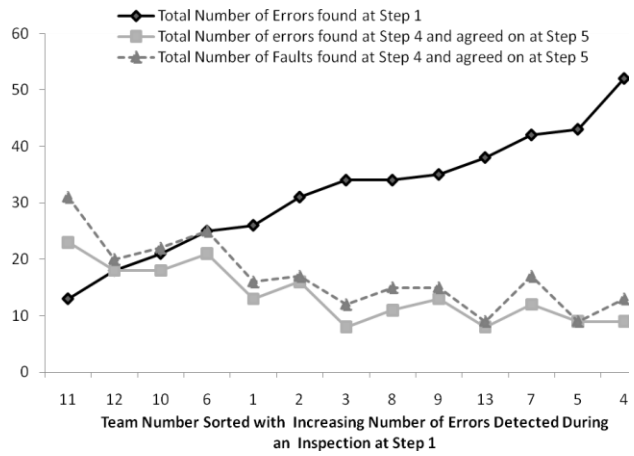

Figure 5. Effect of the Number of Errors Detected Prior to Developing a Requirement Document

ordered by increasing number of errors found at Step 1 (i.e., team 11 found fewest errors and team 4 found most errors at Step 1). Some interesting observations from Fig. 5 are:

- Team 11 found least number of errors at Step 1 and made most errors during the requirements development. Conversely, Team 4 found the most errors at Step 1 made only 9 errors during development (the second lowest number).
- As an exception, Team 12 found the same number of errors (i.e., 18 errors) during an inspection prior to development as well as during the development.

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 show that the errors found at Step 1 had a strong and significant negative relationship with the number of errors made during development in Step 3 ($p < 0.001$; Pearson's R = -0.833; $r^2 = 0.69$) and to the number of faults ($p = 0.001$; Pearson' R = -0.788; $r^2 = 0.62$). 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.

B. Analysis of the Effect of Types of Errors Detected Prior to Developing Requirement Document

While Fig. 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 errors committed during the development.

Fig. 6 breaks down the total errors into the number of *People*, *Process*, and *Documentation Errors* found by each team prior to development and the number of *People*, *Process*, and *Documentation Errors* made during development. The results in Fig. 6 are sorted by the increasing number of each error type found at Step 1 for each team. Some observations from Fig. 6 are:

- During the inspection prior to development, teams found more *People Errors* (an average of 18 errors per document) than *Process or Documentation errors* (an average of 7 errors).
- During development, teams made more *People Errors* (an average of 7 errors per document) than *Process Errors* (an average of 4 errors) and *Documentation Errors* (an average of 3 errors).

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 errors (within that error type) made during the development. The results (in Table II) show that only the *People Errors* found at Step 1 are significantly correlated to the number of *People Errors* made by teams while developing their own documents. This result can be seen in the left-most chart in Fig. 6.

TABLE II. CORRELATION BETWEEN NUMBER OF ERRORS WITHIN EACH TYPE AT STEP 1 AND STEP 3

Errors found prior to the development	Correlation with the errors (within same error types) made during development
People Errors	p= 0.003; Correl. Coeff. = - 0.760, r^2= 0.578
Process Errors	p= 0.059; Correl. Coeff. = - 0.535, r^2= 0.286
Documentation Errors	p= 0.593; Correl. Coeff. = - 0.164, r^2= 0.027

The next analysis examines the effect that the distribution of the errors found in Step 1 (among three error types) had on the overall quality of the documents developed in Step 4. To perform this analysis, we calculated the percentage contribution of errors belonging to each error type found by each team during Step 1. Fig. 7 shows the distribution of errors among three error types that, each team found prior to the development (left side) and during the development

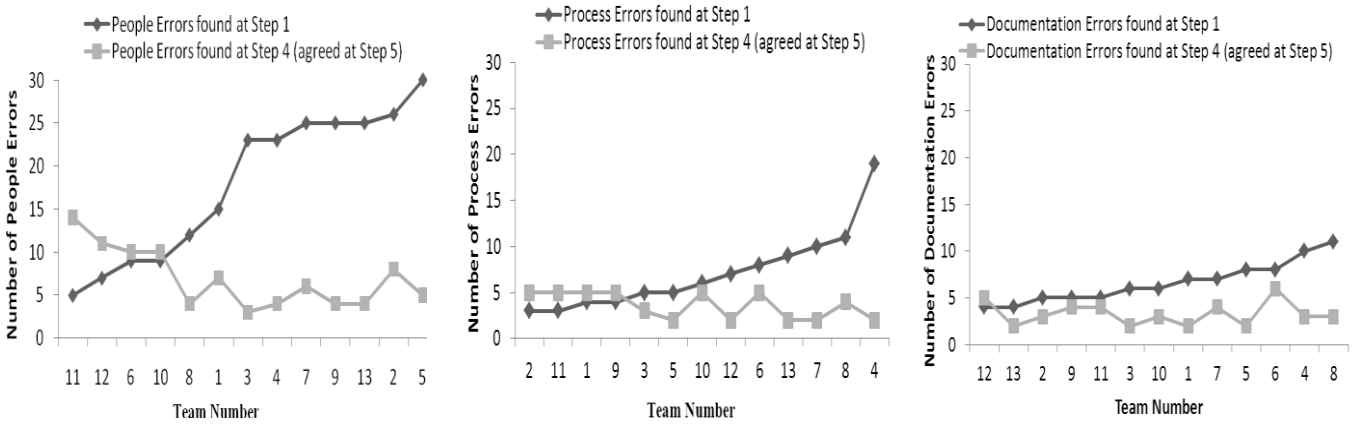

Figure 6. Effect of the Number of People, Process, and Documentation Errors Detected Prior to Developing a Requirement Document

(right side). The teams in Fig. 7 are sorted in increasing order of the number of errors committed during the development of their own document (i.e., team 3 made fewest errors).

An observation from Fig. 7 is that the teams (e.g., 10, 12, 6, 11) that found errors uniformly distributed over three error types prior to development, made more errors during development than the teams (e.g., 3, 13, 5, 7, 9) that found a larger number of *People Errors* prior to the development. This observation led us to investigate if distribution of errors among three error types at Step 1 had an effect on the total number of errors made during the development.

To test hypothesis 3, a chi-square test was conducted to determine if the distribution of errors was significantly different from uniform. The teams that found significantly more number of people errors are highlighted with an arrow in Fig. 7. The rest of teams had a uniform distribution of error types. Fig. 7 shows that, most (but not all) teams that find significantly larger number of people errors (as opposed to more uniform distribution of errors among three error types) prior to the development made fewer total errors while developing their own requirements document. The results from linear regression also confirmed that finding larger number of people errors prior to the development was inversely correlated (at a significant level) to the number of total errors made during the development.

Based on the results, *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.

C. *Analysis of the Distribution of Errors across Fourteen Detailed Error Classes*

To drill down one level deeper, this section focuses on the detailed error classes rather than the three high-level error types. This section analyzes whether the developers in each team made the same type of errors (based on the 14 detailed error classes) during development that they found during the inspection prior to the development.

The comparison of the number of errors in each of the 14 detailed error classes (i.e., 6 people error classes, 5 process error classes, and 3 documentation error classes) found prior to the development to the number of errors made during development was performed for each of the 13 teams. Fig. 8 shows the number of errors within each error class found at Step 1 (lower part of bar) and made at Step 4 (upper part of bar). Due to space constraints, Fig. 8 only shows the results from this analysis for a subset (6 out of 13) of teams. These six teams represent a mix of teams with varying number of the errors made during the development of their documents and are representative of the 13 teams.

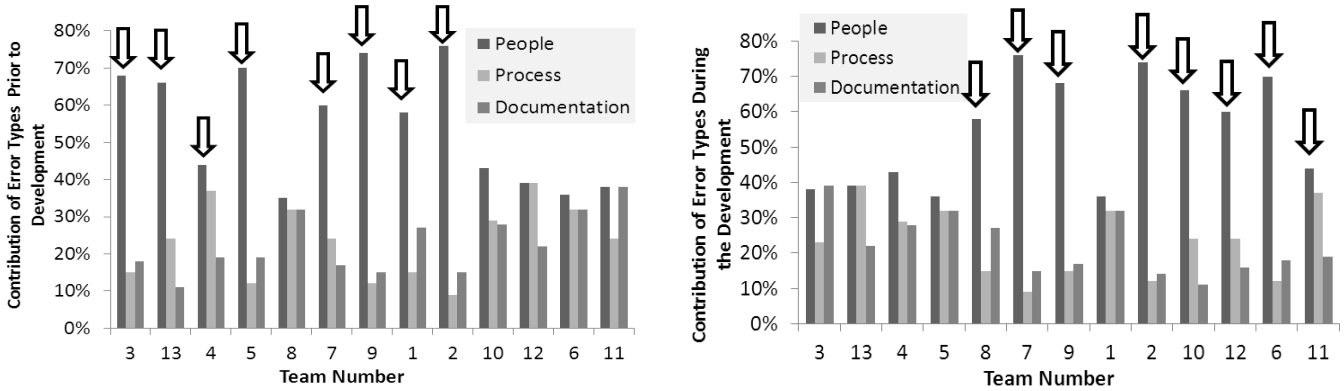

Figure 7. Profile of Error Types Detected Found Step 1 and Errors Made During Development for Each Student Team

Some of the observations from Fig. 8 are:

- Among the *people errors*, errors due to lack of communication, domain knowledge, understanding about specific aspects of problem domain, and process execution errors were commonly occurring. Among the *process errors*, requirement analysis errors are most common. Among the *documentation errors*, requirement organization and specification are the most common.
- The results show that teams made some errors while developing their own documents that they did not find prior to the development and avoided other errors that they had found prior to the development.
- A common trend from the analysis of the results of all 13 teams is that finding more of errors in an error class during Step 1 reduced the likelihood of making them during the subsequent development. This observation is particularly true for *Communication Errors*, *Specific Application* errors, *Process Execution*, *Requirement Analysis*, and *Requirement Organization* errors.
- Even though participants found many *Specification errors* during the Step 1 inspection, they still made a large number of them during the development.

These observations indicate that in some cases we need to develop more formal techniques for error prevention, especially for those error classes that developers made even though they found them in the Step 1 inspection.

D. Analysis of the Effect of Pre-test on the Number of Errors Committed During the Requirement Development

The number of errors correctly classified during the pre-test at Step 1 (see Section B.1.c) was analyzed to determine whether it was related to the number of errors found during 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 ($p = 0.061$; Pearson's $R = 0.532$, $r^2 = 0.283$). However, the correlation was not significant.

V. THREATS TO VALIDITY

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. There is also a threat caused by the lack of a control group. We also plan to address this threat in a future study. 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.

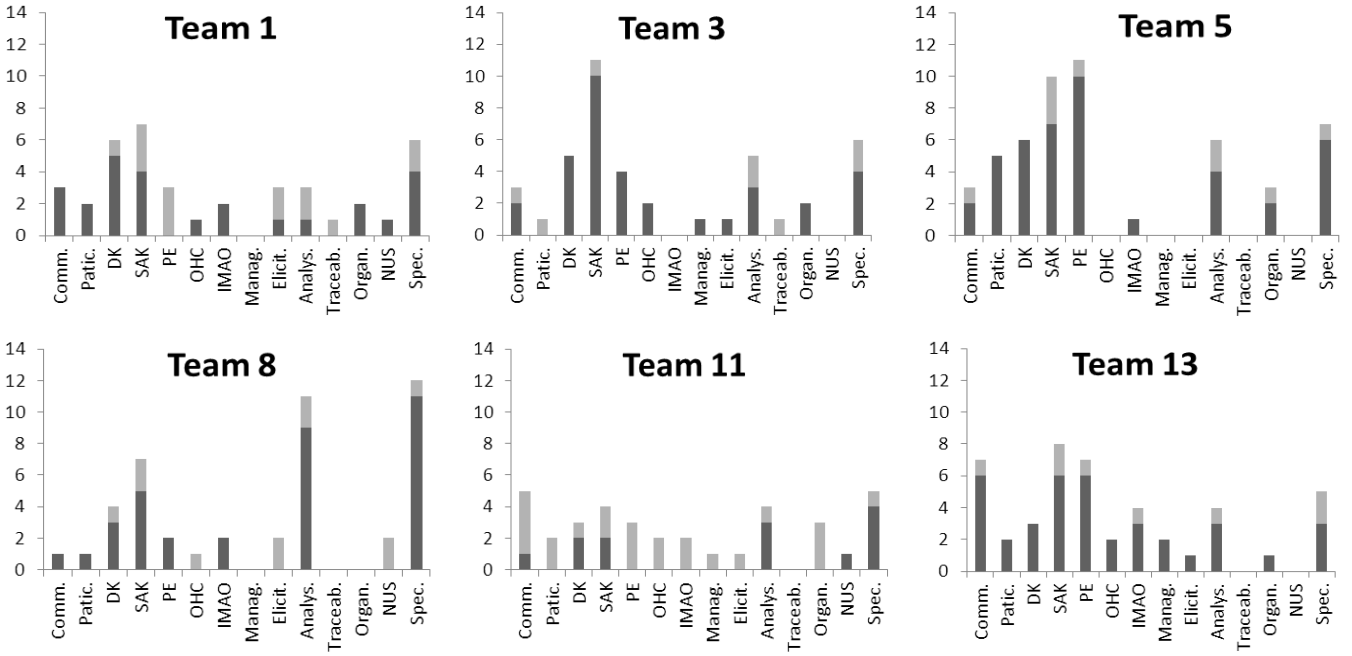

Figure 8. Distribution of People, Process and Documentation Errors Detected at Step 1 into 14 Detail Error Classes

VI. SUMMARY AND DISCUSSION OF RESULTS

This section summarizes and discusses the results from Section IV in light of research questions.

A. Answering Research Questions and Related Hypotheses

A major focus of this study is to investigate the usefulness of the error taxonomy as a defect prevention technique. Discussion of the results related to each of the three research questions follows.

1) ***RQ 1****: Does knowledge about the types of errors that may occur during requirement development make developers less likely to make those errors?*

The first question focused on understanding whether the participants made fewer errors and faults in their own documents after they had found them in the earlier inspection of someone else's document. The results from Section IV.A (Fig. 5) showed that the performance (of participants in each team) during an 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 results was tested statistically there was a significant and strong negative correlation between the two variables. This results thus supports Hypothesis 1:

> ***Hypothesis 1****: The teams whose members find more errors during the inspection of someone else's requirement document will make fewer errors when creating their own document.*

Therefore, using error taxonomy to guide an inspection prior to development of a requirement document is beneficial.

2) ***RQ 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?*

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 commiteed within each error type (Hypothesis 2), and the total number of errors committed (Hypothesis 3) 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.

> ***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.*

The results from Section IV.B (Fig. 6 and Fig. 7) 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.

To gain more insights into the effect of the three high-level error classes, the result from Section IV.C (Fig. 8) compares the errors (within 14 detailed error classes) found prior to the development to the errors made during development. While the the teams completely avoided committing errors within some of the error classes that they had earlier found, they still made other errors that they had earlier found during inspection. Also, the results provided insights into the common errors within each error type.

Since this was initial study and provided positive results regarding the ability of error taxonomy to prevent defects, we will develop more formal techniques to help developers focus on more commonly occurring and redundant errors.

3) ***RQ 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 as well as an accurate predictor of their ability to make fewer errors when developing their own document?*

This question investigated whether the students understanding of the requirement error taxonomy on a pre-test can be used to gauge their likelihood of finding more errors during an inspection prior to the development and their likelihood of making fewer errors during the development of their own document.

The results from Section IV.D show that the average number of errors correctly classified by the team members on the pre-test was 1) positively correlated (though not significantly) to the number of errors found by the members of each team prior to the development, and 2) positively correlated (again not significant) to the number of errors made by each team in their documents. This result indicates that creating software teams that have a better understanding of error classes (using their results from pre-test) can create artifact of higher quality because they make fewer errors.

A major factor that could not be controlled in this study (and might have affected some of the results) was that the participants who evaluated the requirement documents developed in the class were different from the developers of those documents. Because the inspection team members were not necessarily experts in the domain, they might not have found all of the errors and faults that were present in the document. In addition, the ability and motivation of the inspectors to find faults and errors could not be controlled. To minimize the effect of this factor, the four inspectors were randomly selected with the constraint that they come

from four different development teams. This approach should have balanced any effects that could have been caused by any individual inspector or by any events that occurred within a development team.

B. Relevance to Software Organizations

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.

VII. 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.

Acknowledgment

We thank the students in System Analysis & Design course at North Dakota State University for participating in this experiment. We acknowledge the Empirical Software Engineering group at North Dakota State University for providing useful feedback on the design of the experiment.

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