1-1-2006

Empirical Validation Of Requirement Error Abstraction And Classification: A Multidisciplinary Approach

Gursimran Singh Walia

EMPIRICAL VALIDATION OF REQUIREMENT ERROR ABSTRACTION AND CLASSIFICATION: A MULTIDISCIPLINARY APPROACH

By

Gursimran Singh Walia

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

Mississippi State, Mississippi

August 2006

EMPIRICAL VALIDATION OF REQUIREMENT ERROR ABSTRACTION AND CLASSIFICATION: A MULTIDISCIPLINARY APPROACH

By

Gursimran Singh Walia

Approved:

Jeffrey C. Carver Assistant Professor of Computer Science And Engineering (Major Professor)	Edward B. Allen Associate Professor of Computer Science and Engineering, and Grauate Coordinat--or, Department of Computer Science and Engineering (Committee Member)
Ray Vaughn Professor of Computer Science and Engineering (Committee Member)	Dr. Roger King Associate Dean of James Worth Bagley College of Engineering.

Name: Gursimran Singh Walia

Date of Degree: August 5, 2006

Institution: Mississippi State University

Major Field: Computer Science

Major Professor: Dr. Jeffrey Carver

Title of Study: EMPIRICAL VALIDATION OF REQUIREMENT ERROR ABSTRACTION AND VERIFICATION: A MULTIDISCIPLINARY APPROACH

Pages in Study: 96

Candidate for Degree of Master of Science

Software quality and reliability is a primary concern for successful development organizations. Over the years, researchers have focused on monitoring and controlling quality throughout the software process by helping developers to detect as many faults as possible using different fault based techniques. This thesis analyzed the software quality problem from a different perspective by taking a step back from faults to abstract the fundamental causes of faults. The first step in this direction is developing a process of abstracting errors from faults throughout the software process. I have described the error abstraction process (EAP) and used it to develop error taxonomy for the requirement stage. This thesis presents the results of a study, which uses techniques based on an error abstraction process and investigates its application to requirement documents. The initial results show promise and provide some useful insights. These results are important for our further investigation.

DEDICATION

To my family.

ACKNOWLEDGMENTS

I want to thank Dr. Jeffrey Carver for guiding me through research and writing the thesis document. It has been a great learning experience for me. I would like to thank Dr. Thomas Philip for helping me run the experiment study. I would like to thank all the members of the ESE group for providing valuable feedback and suggestions. Also, I want to thank all the students that participated in the experiment run.

TABLE OF CONTENTS

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

This thesis presents research that contributes to the software engineering community in following ways:

1. Devising a formal process (i.e., error abstraction process) that utilizes error information to develop techniques for improving the quality of software products. The "Error Abstraction Process" (EAP) is described in detail, and is evaluated in the context of the requirement phase. The EAP provides a productive (effective and efficient) method of developing an requirement error taxonomy (RET) thereby, improves quality of requirements document.
2. Empirical validation of EAP and RET as a potential quality improvement approach (QIA), and motivation for deriving effective tools and methods using RET along with using EAP throughout the software process.

This thesis presents the EAP and the RET along with experimental evaluation. Initial investigation shows that our proposed approach shows a lot of promise and provides lessons for improving future experiments and investigation. Section 1.1 describes motivation for developing and using EAP, by describing the problem we want to solve. Section 1.2 describes roposed approach, providing an overview of EAP and its scope. Section 1.3 outlines research tasks and activities performed.

Chapter 2 discusses the validation strategy for the EAP and RET in the requirements phase. Chapter 3 provides the related work, and Chapter 4 describes in detail the methods developed. Chapter 5 describes the study design, data collected, and the analysis procedure. Chapter 6 evaluates the research questions, and illustrates the limits of the experimental design. Chapter 7 provides an evaluation of hypotheses, contribution to research and practice communities along with motivation for future research activities and the publications plan.

1.1 Statement of the Problem

Software quality and reliability are major concerns for successful development organizations. Because of the importance of software quality, research in software quality, reliability, measurement, and modeling is important. The software quality problem has been extensively discussed in the literature. Empirical evidence shows that the quality of software products developed is far below ideal [10, 18, 62]. The reasons for this unhealthy condition of software quality include low productivity among developers, haphazard development, non-availability of validated tools/methods for effective software development, and lack of a complete verification process as the product proceeds through the software development process.

Over the years, researchers have developed and evaluated the usefulness of different quality improvement approaches through experiments in controlled and real settings e.g. [5, 6, 7, 12, 54, 57]. The main focus of their research has been on extracting semantic information from faults to allow developers to locate defects early and improve software quality.

Researchers have also examined the cause-effect relationship of faults and their impact at different phases of the software lifecycle. In addition, different fault classification taxonomies have been developed and evaluated. More details on the various quality improvement approaches using fault classifications are provided in Section 3.3.

While these fault classifications have proven beneficial, defects still exist. There is still room for additional methods to fill in the remaining gaps for more complete and sound verification process. Quantifying, classifying, and locating an individual fault is a subjective and intricate notion, especially during the requirements phase [6, 31]. In addition, some of the existent fault classification taxonomies have been criticized for their inability to satisfy certain attributes (e.g., simplicity, comprehensiveness, exclusiveness, intuitiveness etc) [16, 33]. These fault classes have also been used in quality measurement and improvement frameworks [14, 24, 54, 57]. However, because the fault information fails to target the underlying cause, these frameworks were not as effective as they could have been. We believe that focusing only on faults, that is, mistakes that are recorded in a document, will not eliminate the underlying mistakes that can lead to multiple faults and failures.

A similar notion is the quality improvement approach called “Root Cause Analysis” (RCA), which focuses on analyzing the causes of faults. However, due to the expenses incurred, the substantial investment of resources, and the large number of actions that result, RCA did not find widespread success [19]. Building on this idea, the Orthogonal Defect Classification (ODC) was used to enable in-process feedback to developers.

However, the ODC also focuses on using semantic information from only the faults to extract a cause-effect relationship in the development process [16].

In this thesis, our goal is to push RCA further by extracting error signatures in a similar fashion as done by ODC to help developers get to the root of a fault more quickly and to understand the real problems in software development. Stated more formally, my goal is (as follows):

- To analyze faults to determine the underlying causes (i.e., errors) and to provide developers with methods for identifying and classifying the errors and using those methods to improve software quality.

1.2 Proposed Solution

1.2.1 Definitions

Throughout this thesis, I use the following terms:

- Error: A mistake made in the human thought process while understanding the information provided, solving the problem or using methods and tools [28].
- Fault: A concrete manifestation of an error in a software artifact. One error may cause several faults and multiple errors may result in the same fault [28].
- Failure: Execution of a fault. It is a departure of the operational software system behavior from user expected requirements. Again, the same failure may be caused by multiple faults, and some faults may never cause a failure [28].

- Error Abstraction: The process of examining a fault to determine the underlying misunderstanding (error) that caused the fault. This definition is adapted from initial work done on the process of error abstraction by Lanubile et al. [31].
- Human Reasons: Factors related to cognition and human thought process that are not unique to software engineering.

1.2.2 Importance of Focusing on Errors

This point of this is that addressing the quality problem by using error information will provide better results than by using only fault information.

To that end, previous research by Lanubile, et al. has provided evidence that using an error abstraction process in analyzing a requirements document may be a useful approach [31]. This thesis builds on original work by quantifying the process of error abstraction with an error classification taxonomy that contains information about the cause effect relationship between errors and software quality.

Previous research has shown that the software engineering literature is not sufficient to address all the errors that can occur throughout the software development. Case studies report that human reasons also contribute to fault injection in software development [13, 35, 47, 62]. Since software development is a human-based activity, it is useful to investigate various phenomenon associated with the human mental process and its fallibilities. Thus, one challenge is integrating the relevant contribution from human cognition with the error abstraction process.

1.2.3 *Proposed Approach*

This thesis describes a quality improvement approach (QIA) for improving quality of software products at each stage. For that reason, I described the generic process of applying EAP that could be used at any development stage for developing quality improvement techniques. Application of EAP to a software development phase requires analyzing faults and abstracting as many kinds of errors in a particular phase that can lead to faults and failures and developing error taxonomy. I have applied EAP to requirements phase and developed "Requirement Error Taxonomy" (RET). The detailed description of the process of developing the RET using the EAP is described in Section 4.2.

Also, I describe the process of using QIA (i.e., EAP and RET) to improve the quality of requirements document during an experiment run. I have investigated the QIA (EAP and RET) to analyze requirements document. The process of error abstraction consists of analyzing the nature of faults and determining the underlying mistakes, or errors, responsible for those faults. We train inspectors on how to abstract errors from the faults to help them understand their actual mistakes in a similar way as done in an earlier study [31]. Multiple faults may be abstracted to different underlying mistakes or to the same mistake depending on the nature of the information involved in those faults. In addition, a single error can lead to multiple faults. So, understanding the errors that occurred when creating the requirements document can lead the inspector to locate additional faults, related to those errors, in the requirements document.

However, during the process of abstracting errors, there is the potential that all errors may not have been located. Thus, to augment the initial error abstraction work [31],

the requirement error taxonomy (RET) will help inspectors do a more thorough job of locating errors and their corresponding faults. It is likely that with an understanding of errors, developers may see additional faults. The purpose is to see whether using RET in EAP helps developers efficiently reach actual causes of faults. This step is described in detail in Chapter 5.

In this thesis, the primary aim is to investigate the application of QIA at the requirement phase. Also, another purpose is to evaluate and improve the EAP before using it for subsequent phases.

Also, we want to evaluate and improve the RET before using it to derive any tools and methods, which is a research task for the future.

1.3 Research Tasks

The research activities are shown in Figure 1. The first task consists of designing a generic error framework that can be used for all development stages (e.g., requirement, design, coding, testing, etc.). The second task consists of application of EAP to the requirement phase and developing requirement error taxonomy (RET). This thesis explores the usefulness of the QIA in the requirements phase. The third task consists of validating the EAP as a process of developing quality improvement techniques, as well as the QIA for its intended task.

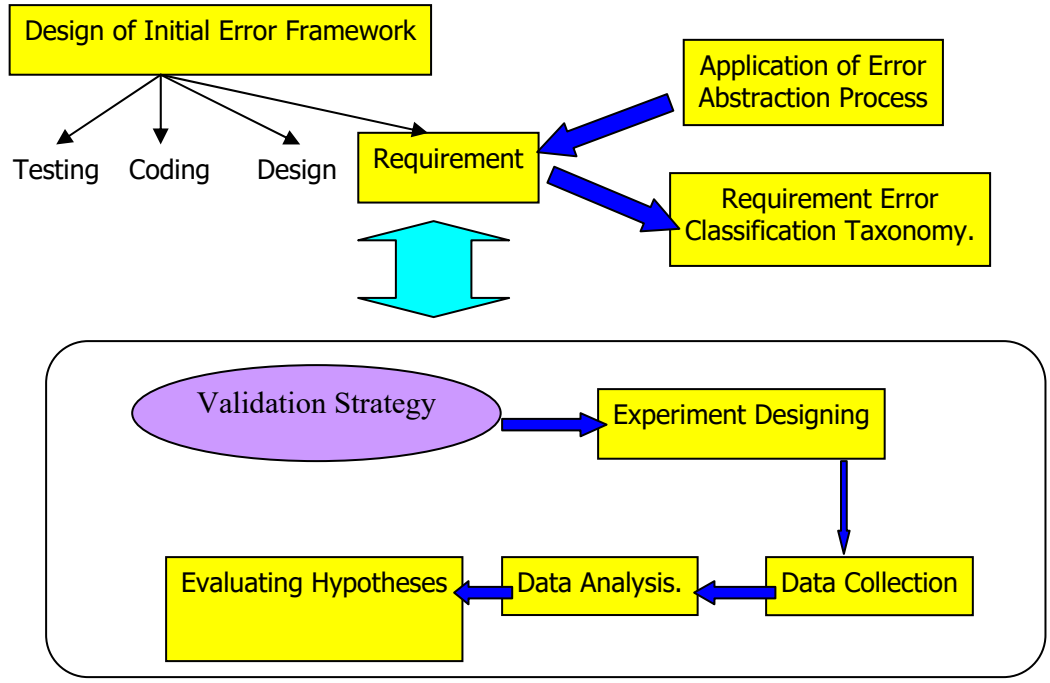


Figure 1: Research Tasks

CHAPTER II

VALIDATION STRATEGY

Validating the proposed solution presented in Chapter 1 requires evaluation at two levels as described in Section 1.2.3.

The first is evaluating the effectiveness of EAP as a process for developing quality improvement approaches (e.g., RET). Evaluating EAP as development mechanisms requires determining the potential of the development process (i.e., EAP), its constituents, and the solution provided by EAP (i.e., RET) in context of the problem statement, identified in Section 1.1. This evaluation considers whether this research is making useful contribution to the problem area it addresses. The validation strategy for EAP is described in Section 2.1.

The second is evaluating the use of EAP and RET by developers in terms of productivity, insights provided, and effectiveness regarding the intended task and environment. Also, it requires evaluating the RET by characterizing its essential attributes and the task to which it has been tailored. The evaluation criterion for this technique is described in form of "Research Questions" in Section 2.2.

2.1 Validating Error Abstraction Process (EAP): Development Process

The validation strategy consist of forming high level hypothesis, setting GQM goals, and listing evaluation questions or research questions, followed by mapping the metrics used for research questions.

2.1.1 Hypothesis

H1: The "Error Abstraction Process" (EAP) is a feasible way of creating effective quality improvement techniques that will provide an improvement over approaches based on faults and other fault based techniques.

2.1.2 GQM goals

G1: *Analyze* the literature, fault modes, fault classifications, historic data, and multidisciplinary survey *for the purpose of* characterizing *with respect to* underlying mistakes from the *point of view of* software developers and researchers *in the context of* the requirement phase of the software process.

2.1.3 Research Questions

1. RQ 1: Is EAP a feasible way of developing effective error taxonomies?
    a. Are all the errors and their resulting faults within error classes detectable or feasible?
    b. What is the contribution of research from human cognition, psychology, and similar fields that helps locate more errors and corresponding faults?
2. RQ2: Is EAP a feasible way of improving error taxonomies?

a. Does mapping back to human errors improve the error taxonomy?

b. Are there any errors that can be abstracted from real faults and are not present in the RET?

2.1.4 Metrics

1. M1: Errors and Faults caused by each error class.
2. M2: Contribution of human reasons to error and fault injection.
3. M3: Errors not classified into requirement error classes.

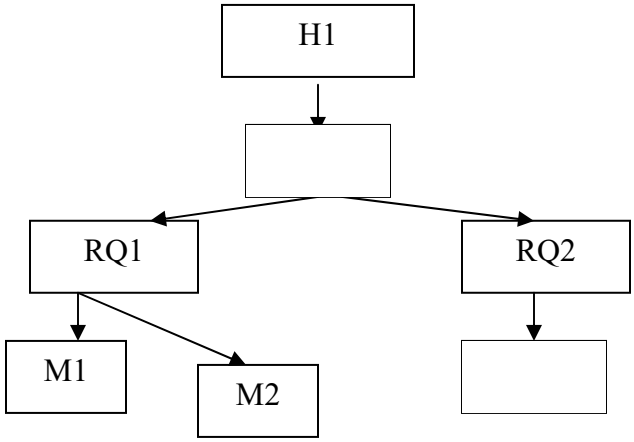


Figure 2: Mapping of Hypothesis, Reseach Questions and Metrics: Validating EAP.

2.2 Validating QIA: EAP and RET

2.2.1 Hypothesis

H2: This QIA improves productivity of developers and software quality over approaches based on fault classifications and similar techniques.

H3: The RET fulfills necessary criterion of essential attributes (i.e., simplicity, understandability, applicability, intuitiveness, orthogonality, comprehensiveness, usefulness and uniformity across products) [33] and is intuitive and useful to software developers.

2.2.2 GQM Goals

G2: *Analyze* literature, error modes, mistakes *for the purpose of* characterizing *with respect to* error types and classes from the *point of view of* researchers *in the context of* the requirement phase.

G3: *Analyze* QIA *for the purpose of* characterizing it *with respect to* effectiveness and efficiency *from the point of view o*f researchers *in the context of* a classroom experiment.

2.2.3 Research Questions

1. RQ3: Does the RET satisfies all essential attributes (i.e, simplicity, understandability, applicability, intuitiveness, orthogonality, comprehensiveness, usefulness and uniformity across products) [33]?
2. RQ4: What is the adequacy of error types and error classes in RET?
    a. Is the description of each error errors adequate and clear?
    b. Are there any errors that are in the wrong class?
3. RQ5: Is the QIA effective for the intended task?
    a. Does the QIA (EAP + RET) improve the productivity of developers?
    b. Does the QIA provide useful insights regarding the causes of low productivity during the requirement phase?

c. Does the QIA accelerate learning and communication, and is useful for developers?

2.2.4 Metrics

These metrics will help us answer these research questions, thereby helping us to evaluate the QIA and RET in particular.

1. M4: Characterization of RET on essential attributes.
2. M5: Subjective feedback from the use of the RET and EAP.
3. M6: Causes (Error types/classes) of redundant, time consuming and multiple faults.
4. M7: Impact of using QIA on improvement in effectiveness (i.e., fault detection density) and efficiency (i.e., fault detection rate) of developers.

3.3 Relevance

Evaluating the EAP and QIA as described in sections 2.2.1 and 2.2.2 will give an indication of their effectiveness in the context of improving quality in the requirements phase. If the evaluation is positive, then this research provides motivation for using the EAP in subsequent life cycle phases and for developing a set of tools and methods to assist developers throughout the development process.

If the evaluation is negative, then there is indication that the problem is not effectively understood. Either the techniques already developed need to be improved or the process (EAP) itself needs to be improved to be re-applied effectively to develop better quality improvement techniques and validate them.

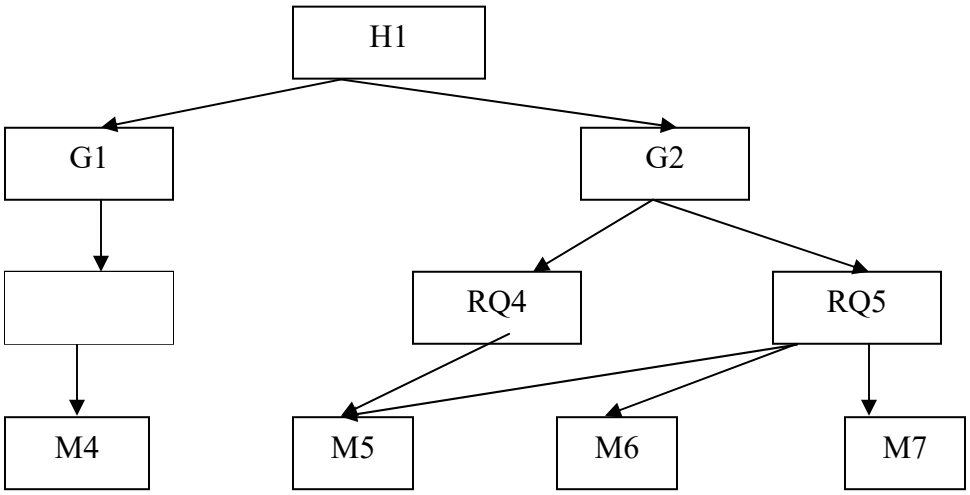


Figure 3: Mapping of Hypothesis, Reseach Questions and Metrics: Validating QIA.

CHAPTER III

RELATED WORK

This chapter describes the literature surveyed in various topics and fields, and how each related work contributes to the work presented in this thesis. This chapter is divided into Sections, with each section describing the related work in different areas. Section 3.1 describes the related work in the software quality problem, followed by review of different research frameworks in Section 3.2. Section 3.3 describes various QIA's and their limitations and drawbacks. Section 3.4 describes the related work on EAP, followed by related work in multidisciplinary fields in Section 3.5.

3.1 Empirical Studies on Software Quality

The problem statement described in Chapter 1 concentrates on the software quality. Before devising an effective solution we must understand the problem. This section describes empirical studies on the quality of software in industry, and the issues regarding the condition of software quality.

A study of 350 companies and 8000 software projects indicated that one-fifth of the projects are cancelled before completion or exceed the estimated resources with only 9% delivered on time and within budget [63]. Another study indicates that 40 to 50% of the effort in current software projects is spent on avoidable rework [10, 18].

A third empirical study reports that if a fault is not fixed at a requirement stage, it becomes two hundred times more expensive to fix after the software is developed [10].

The reasons for this condition of software quality are believed to be low productivity of developers, lack of insights into major problem areas, lack of understanding/usage of effective tools for intended tasks, and lack of a sound verification process, as the product proceeds throughout the software process [18]. Also, inadequate understanding of needs of user and inability to communicate and build consensus among developers is a major issue [10].

These studies help us understand the real issues involved in software quality that includes attention to user needs, modeling effective processes, integrating methods to characterize development process, and validating methods through empirical studies. The result of the solution will be a sound verification process leading to successful development of software products.

3.2 Review of Framework Designs

To develop the research framework design for this thesis work, the analysis of literature about different aspects of software measurements and framework designs is conducted. This is done to understand the important constituents needed for a framework design to measure software problems. Literature provides a lot of information on the aspects of the software measurements. Various framework designs are also present in literature that describes techniques used for quality measurement using counting problems and defects.

Important frameworks reported in literature include Grady, Humphrey, Fenton, and Baumet [24]. This literature describes a "*software measurement environment*" that talks about the basic elements needed to define a software measurement and the entire framework designs need to be consistent with this software measurement environment [24]. Each framework design also describes a rationale as why is it measuring particular thing e.g. defect or fault? Analysis of the important elements of the software measurement environment (e.g., goals and objectives, scope, degree of measurement, etc.) and different frameworks helps to learn the important constituent elements needed to provide the complete and consistent measurements of actual problems. Also, I analyzed the reasons for the shortcomings of the frameworks provided in literature (i.e., focus on active faults) in order to overcome them in my error framework design (i.e., focus on errors and corresponding faults). Each component of the error framework is developed by thorough analysis and contributes to the overall solution in one way or another.

Analyzing this literature helped me to constraint and modularized my framework design into pertinent components. The framework design for our research is described in detail in section 4.1 in terms of its goals, domain of application, and the constituent elements.

3.3 Review of Quality Improvement Approaches

This section discuss various quality improvement approaches (QIA) that were consulted to develop the proposed approach (described in Section 1.2.3) to problem statement (described in Section 1.1), and for using the proposed approach in the proposed framework design for improving software quality (described in Section 4.1).

One of the QIA is NASA software engineering laboratory (SEL) software process improvement (SPI) process that describes mechanisms for SPI to better understand software process, product, and measurement of valuable attributes. These mechanisms include experience factory, Top Down approach, and Bottom Up approach that helped them package experiences for faster learning [57]. This approach uses their defined set of fault classes for analyzing faults from different phases to reduce risk, costs and cycle time. They also conducted experiments to validate the effectiveness of their approaches. I analyzed this QIA and its effectiveness and limitations in context of the problem statement described in Chapter 1.

Another QIA is the measurement framework used by SEI with goal of understanding and predicting software process efficacy and software product quality [24]. This QIA use fault classifications to build checklist for improving product quality. This approach uses report forms and checklists to record and analyze information regarding different faults. Because measuring and locating faults does not tell the whole story about the underlying mistakes, the data collected using this approach does not reveal the underlying causes of the faults. Thus, the framework used in this approach did not facilitate effective learning, because it used fault information to obtain a cause effect relationship in order to understand the actual problems in software development.

Another QIA is called “Root Cause Analysis” (RCA), which focuses on analyzing the causes of faults [19]. However, due to the expenses incurred, the substantial investment of resources, and the large number of actions that result, RCA did not find widespread success. Building on this idea, another similar QIA was “Orthogonal Defect

Classification" (ODC). ODC was used to enable in-process feedback to developers. Here again, the ODC focuses on using semantic information from only the faults to extract a cause-effect relationship in the development process [16].

Similar QIA's involves extracting semantic information from faults to allow developers to locate defects early and to improve software quality. For this reason, various researchers have examined the cause – effect relationship of faults and their impact at different phases of the software lifecycle. In addition, different fault classification taxonomies have been developed and evaluated [5, 6, 7, 12, 14, 16, 43, 53, 54, 61]. Different classification taxonomies were used to measure the healthiness of the product as it proceeds through the development process. While these fault classifications have proven beneficial, they do not offer complete and sound verification process.

Quantifying, classifying, and locating an individual fault is a subjective and intricate notion, especially during the requirements phase. In addition, various fault classification taxonomies have been criticized for their inability to satisfy essential attributes and also using fault classes can not eliminate the underlying mistakes that can lead to multiple faults and failures [10].

Furthermore, these fault classifications have been used to derive various kinds of tools and methods to assist developers [4, 5, 6, 54, 55]. One important tool is reading protocols that guide inspectors and developers to locate faults and fix them early. Researchers have developed and evaluated different reading protocols, namely checklist-based, fault-based, and perspective-based protocols. These techniques are useful for the fault detection process; however, since they are derived from fault classification

taxonomies, they cannot eliminate the underlying mistakes as well. Thus, they can not be fully effective in locating all defects.

3.4 Review of Error Abstraction Process

Reviewing different QIA's described in Section 3.3, indicates that developing techniques based on error information (i.e., the underlying causes of faults can be more effective in eliminating all the defects). To that end, there is empirical evidence that using error information in analyzing a requirements document may be a useful approach [31]. In this research, the author investigated the utility of error abstraction to augment the fault detection density in a requirements document. This research is a major inspiration for our research work that builds on their work to investigate the usage of a more formalized error abstraction process in the context of the requirement phase. The authors in their work did not utilize any formal error detection process, and the approach was based on creativity of developers to abstract errors from fault. Our research work builds on their work to develop a more formalized approach (i.e., error classification taxonomy to improve software quality solution).

3.5 Review of Multidisciplinary Approach

The proposed approach requires the understanding of human fallibilities and its consequences on software quality. This section discusses the research from other fields, and how they contribute to the proposed approach.

Investigating the evidence regarding the contribution of human reasons, I came across a case study in root cause fault analysis that reports a significant influence of

human factors on fault injection in software development process [62]. Also, fault analysis of the software generation process classified some faults as random for which no specific cause could be identified [62]; and I want to analyze whether these faults can be attributed to human reasons. Another work describes that the causes of some faults in software development occurred due to the lack of communication between people participating in the software development [11, 34]. These findings indicate that the survey of errors in human reasoning can make a useful contribution to the error abstraction process. Because there are people involved in development, the human mental process for analyzing actions and consequences relevant to software development must also be considered.

Investigating various modes of human fallibilities, I noticed that various researchers have analyzed human errors from different viewpoints. The principal investigator in this area, James Reason took a psychological viewpoint for classifying human errors based on intentions, actions, behavior, and sequences [47]. He divided errors into mistakes, which are planning errors, and slips or lapses, which happen due to the wrong execution of actions. He used the GEMS (General Error Modeling System) to model various performance levels to decompose mistakes as knowledge-based or rule based [45]. I analyzed the Reason's research to understand human mental process and its fallibilities. In another work, Allistair Sutcliffe and Julia Galliers also analyzed human errors and developed human error classification [3, 4]. I analyzed the human errors in this classification to see whether they can really constitute an error in the requirement phase

and where it fits in my error classification. I also looked at the impact of these errors on requirements document.

Another relevant taxonomy is the Human Factor Analysis and Classification System [21] based on Reason's Swiss-Cheese model [47]. It traces back from active faults and failures to the underlying mistakes through different levels of mistakes and the human errors at each level. It is relevant to my work, as it describes various levels of hidden mistakes or errors that lead to faults and failures. HFACS was used in aviation industry to increase air safety [21]. I analyzed this classification to locate the mistakes at a level compatible with other errors in my error classification. I analyzed errors that constitute requirement errors and can lead to faults during the software process. I analyzed what faults can be attributed to these errors in the requirement phase.

Furthermore, Norman used Reason's idea of inappropriate action execution on the side of the individual to describe a theory of action [42]. Norman analyzed classes of errors that people make while interacting with the system. Norman's framework was used for minimizing occurrence of errors and its effect during user interaction activities. Rasmussen also described his Decision Ladder Model [45] that models the decision making process using various states in decision making cycle, and is a human performance model. These classifications were basic reflections of human mistakes in different circumstances. I also analyzed them to bolster my error classification. Also, Norman used his background in cognitive science to discuss slips as capture errors, description errors, data driven errors, associative action errors, loss of activation errors and mode errors etc. I also analyzed these errors to find whether they can really constitute

errors in my requirement error taxonomy. Another study outlines the Human Error Identification (HEI) tool for analysis of cognitive errors in ATC during planning and decision making. They described three main types of taxonomies describing context of error, production of error, and recovery of error [58]. This taxonomy was analyzed to find the human errors in planning and specification stage of the requirement phase that can be attributable to human reasons.

CHAPTER IV

METHODS DEVELOPED

This chapter describes the solution devised for the problem, describing the methods developed to help developers locate errors and their corresponding faults at its source of origin. Section 4.1 describes the "Error Framework Design" to guide the research investigation for this thesis work and for future investigation. This framework design is developed with the aim to provide correct, complete and consistent measurement of the errors across different phases. Section 4.2 describes the EAP as a process of developing error classification taxonomy. Section 4.3 describes the technique developed i.e "Requirement Error Classification Taxonomy" in detail. Finally, section 4.4 describes the nature of future investigation (i.e., the family of techniques that will be developed using the RET).

4.1 Error Framework Design

The error framework describes the mechanism of using the error abstraction process to extract error signatures, thereby helping developers get to the root of faults efficiently and understand actual problem in software development, which can improve software quality. The framework is described in terms of its scope (Section 4.1.1), goals and objectives (Section 4.1.2), and its components (Section 4.1.3). "Error Framework Design" is shown in Figure 4.

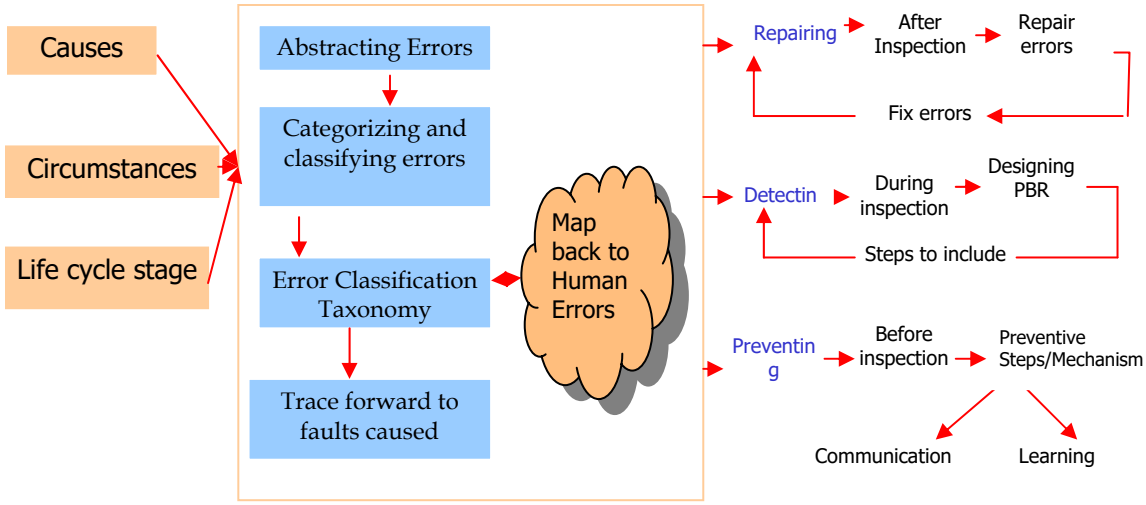


Figure 4: Error Framework Design

4.1.1 Scope of Framework Design

The Error Framework Design has domain of investigation that concentrates on the following issues in the problem identified:

- Description of attributes to help developers measure, learn, and communicate major problems in software development.
- Implementation of error abstraction process and using it to improve quality and gain useful insights into the development process.
- Assisting developers by developing an array of effective techniques and tools for intended tasks.

4.1.2 Goals and Objectives of Framework Design

The objectives that the framework intends to achieve are (as follows):

- Able to provide sound verification process for improving the quality of software produced.
- Track and verify the progress of product throughout the software development process.
- Improving productivity of developers by providing them with effective techniques and the intended tasks and environment they are best suited for.
- Accelerating learning for developers and provide useful insights into major problem areas in software development.

4.1.3 Components of Framework Design

The whole framework design consists of three different components (as follows):

- "*Measurable Attributes*" describe the cause of faults, the circumstances under which the error occurs, and the development stage in which the error occurs. These attributes help develop a pattern of occurrence to improve learning and communication among developers. As the name suggests, these attributes are easily measurable and provide useful information to enhance developers' understanding of actual problems.
- Implementing "*Error Abstraction Process*" (EAP) requires abstracting errors and developing techniques for helping developers to locate errors and corresponding

faults. The process of using EAP to develop technique is described in detail in Section 4.2, and the technique developed is described in detail in Section 4.3.

- The "*Methods and Tools*" are developed using the output from implementing EAP (i.e., error classification). This component is the topic of future investigation, and is included here only for the sake of being thorough. The complete framework of family of methods and tools are described in section Section 4.4.

4.2 Error Abstraction Process

Application of EAP to a software development phase requires analyzing and abstracting as many kinds of errors in a particular phase that can lead to faults and failures. To do this abstraction, one needs to survey relevant information in different sources including software engineering literature, fault classification taxonomies, and historic data from previous projects. An additional important source that needs to be used, to incorporate the need to understand the contribution of human cognition errors, is research from fields like human cognition, psychology, and failure management. Analyzing human error taxonomies and other relevant human cognition related fallibilities helps to make the list of abstracted requirement errors as comprehensive as possible.

The next step of this proposed approach is to group together similar types of requirement errors from the list of abstracted errors to develop an "Error Classification Taxonomy". The error classification taxonomy will explain similar kinds of errors grouped together to help developers understand the basic symptoms of particular kinds of

errors. The aim is to develop a simple error taxonomy with orthogonal classes that is still comprehensive and intuitive.

The next step is to map the error taxonomy back to the human error taxonomies (described in Section 3.5) to ensure completeness and provide additional confidence. The error taxonomy describes each error in detail, and traces forward the impact the error is likely to have on software quality in terms of faults that can result from it. The error taxonomy aims to aid developers by providing a comprehensive list of errors classified in such a way that helps them check different errors and their corresponding faults at their sources, thereby reducing the rework effort.

In this thesis, we are investigating the application of our proposed approach at the requirement phase. So, the EAP is applied at the requirement stage to develop the requirement error taxonomy (RET) discussed in Section 4.3. The whole process of error abstraction is described in Figure 5.

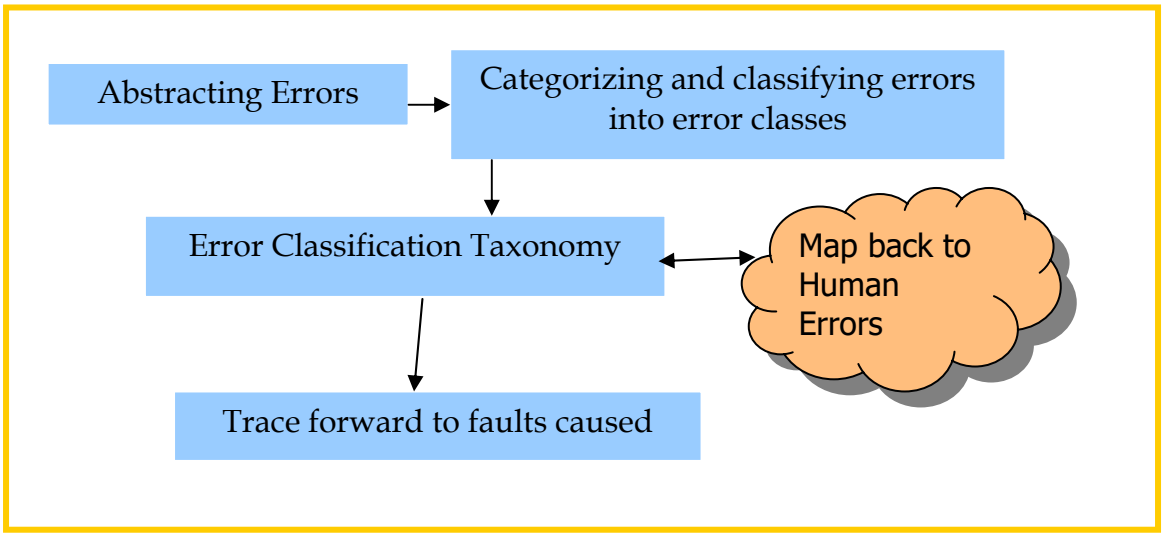


Figure 5: EAP: A Process of Developing Error Classification Taxonomy

4.3 Requirement Error Classification Taxonomy

This section describes in detail the RET, and its impact on software quality in the terms of faults that are likely to be caused. The RET groups errors into 3 types (i.e., People errors, Process errors, and Documentation errors), and each error type is further refined into more detailed classes as shown in Figure 6. Each error class contains similar errors grouped together to help developers understand the symptoms of that error class. Further, the RET also explains the faults likely to be caused by errors in each error class.

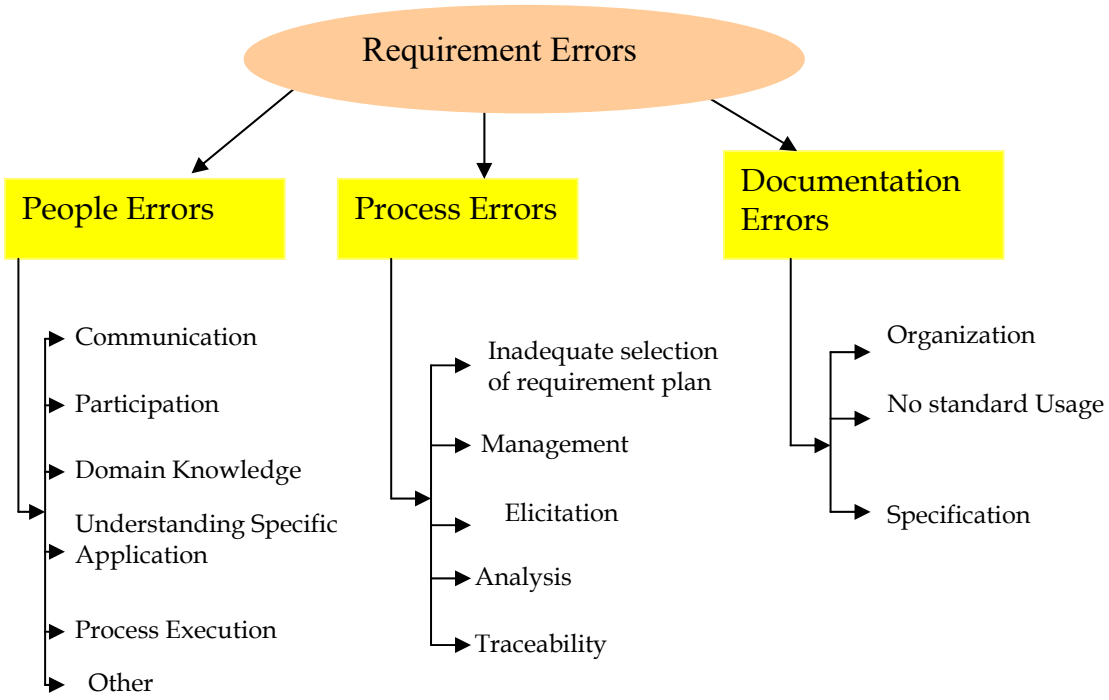


Figure 6: Requirement Error Classification Taxonomy: Error Types and Error Classes.

For each error class, the RET provides a description and example of the error as well as a fault that could result. To illustrate the errors and corresponding faults, we use the following example application systems:

- Automated Teller Machine Network (ATM): This system supports a computerized banking network. The ATM network does not work independently. It works together with a bank computer. There are clearly defined interfaces for the different systems.
- Automated Ambulance Dispatch System (AAD): This system supports the computer-aided dispatch of ambulances. The goal of the system is to improve utilization of ambulances and resources. Also, the systems should reduce the average time to respond to emergency incidents by improving dispatch decisions based on recommendations by the system
- Parking Garage Control System (PGCS): The PGCS controls and supervises the entries and exits of a parking garage. The system allows or rejects entries into the parking garage based on the number of available parking spaces. The system handles both monthly ticket parking and daily ticket parking.

4.3.1 People Errors

These errors concern individual fallibilities and are caused by the people involved in the project development. These errors arise from various kinds of mistakes, misunderstandings or forgetfulness (e.g., lack of communication, misunderstanding of

customer needs, misunderstanding of the specific application and other similar errors). Following are various errors classes within this error type.

A) Communication Errors

- Lack of communication among stakeholders including communication with customers/users within the team and between teams.
- Unclear lines of communication and authority leading to lack of consensus on technical standards and approaches with teams.
- Lack of communication of changes made to a document.

Example

- Error: Customers do not communicate all requirements (e.g., an ATM system should dispense different types of currency such as US dollars, Euros, etc).
- Fault: Omitted functionality (i.e., The ATM only deals with dollars and capability to dispense other currencies is missing).

B) Participation Errors

- Lack of participation of all stakeholders (including all user groups) during development.
- Lack of motivation or rivalry among the parties involved in the project development.
- Lack of an arena or facilities to mediate conflicts between stakeholders.

Example

- Error: A specific user (e.g., the bank manager) was not involved in the requirements process; and therefore, his needs have not been included (e.g, an ATM system should support multiple copies of a cash card simultaneously).
- Fault**:** Omitted or incorrect functionality (i.e., the SRS may not specify the use of multiple cards).

C) Domain Knowledge Errors

- Lack of experience or domain knowledge of the requirements author.
- Misunderstandings due to complex nature of task domain.
- Lack of skills required for performing a particular task (e.g., person involved in requirement engineering does not posses necessary knowledge and is told to create the task plan).
- Some properties of the problem space are not investigated leading to incorrect dependencies between pieces, wrong assumptions, or incorrect behavior.
- Mistaken assumptions regarding the states, preconditions, and post-conditions.

Example

- Error: The requirements author is not knowledgeable in a particular area about which he must specify requirements (e.g., for an ambulance dispatch system the requirements author does not understand medical incidents, but must specify this information in the requirements)
- Fault: Incorrect functionality (i.e., incorrect algorithm for assignment of ambulances may be specified in requirements).

D) Understanding Specific Application Errors These errors are caused due to the misunderstandings of the specific applications (as opposed to the general domain):

- Mistakes regarding the expression of end state, output, goals, or objectives to be achieved.
- Misunderstanding or mistakes in resolving conflicts (e.g., there are unresolved requirements or bad requirements and are agreed on by all the parties).
- Misunderstandings regarding the relationships and dependencies between individual pieces and the real world.
- Mistakes or misunderstandings regarding the constraints of timing or timing relationships between commands in concurrent execution of process.
- Misunderstandings of the data dependency constraints or conflicts regarding the restrictions on order of command when 2/more processes access the same input.
- Misunderstandings of ordering of events/commands or functional properties.
- Misunderstandings of the software interfaces with the rest of the system and hardware interfaces.
- Mistakes or misunderstandings in mapping of the inputs- outputs, input space-processes, and process-output.

Example

- Error: Misunderstandings regarding the ordering of the events (e.g., in AAD, for providing “ambulance service”; a precondition such as setting ambulance status data to current is specified after incident is completed).

- Fault: Wrong precedence relationship (i.e., wrong ordering of the events or commands in providing "Ambulance Service" may be specified in requirements).

E) Execution of process Errors

- Mistakes in applying the process, irrespective of its adequacy to the task at hand.
- Execution/storage mistakes, disordering of steps and lapses on the part of people executing the method.

Example

- Error: Mistakes in application of a particular process (e.g., while applying the traceability process, some traces are disordered or left out).
- Fault: Missing traces and dependencies.

F) Other human cognition errors

- Mistakes caused by adverse mental states, mental fatigue, loss of situation awareness, lack of motivation, or task saturation.
- Mistakes caused by environmental conditions (e.g., temperature, lightning, etc.).

4.3.2 Process Errors

These errors are caused by mistakes in selecting the means of achieving goals or objectives. The errors are due to the inadequacy of the processes employed during the requirement-engineering phase. The various methods/processes include the planning - management- elicitation and other processes. Different kinds of errors can occur in these processes.

A) Inadequate methods of achieving goals/objectives

- Missing or inadequate setting of goals and objectives.
- Wrong method chosen because some system-specific information was omitted or misunderstood.
- Selection of an existing successful method for completely unknown situation without any investigation.
- All the important facts were understood, but the wrong method was chosen.

Example

- Error: Mistake (wrong information used) in selecting a plan (e.g., for selecting a plan of "issuing warnings" in AAD, the planner uses the tolerance values of status data for selecting a plan; however, the information used should be the tolerance values of the open incidents).
- Fault: Incorrect processes or unperformable processes (i.e., "Issuing Warning" functionality cannot be achieved with specified information).

B) Management Process

- Omissions or misunderstandings regarding the assignment of resources to different development tasks.
- Inability to provide leadership and necessary motivation.
- Omission or misunderstanding of all the alternatives and their impact.

Example

- Error: Mistakes in assignment of resources (e.g., in automated dispatch system, in case of "review incidents," the resources are not allocated for batching up incident reports and receiving call from incident sites).

- Fault**:** Incompleteness and ambiguity in requirements due to unavailability of the resources may be specified.

C) Weak Requirements Elicitation Process

- Lack of questionnaires or interviews for eliciting requirements from customer.
- Slips or lapses or lack of awareness of all sources of the requirements.
- Inadequate procedure for collecting requirements from these various sources relevant to the problem domain.

Example

- Error: Inability to elicit all the requirements (e.g., in LAS system, the developers are not able to elicit the requirements regarding the response time to emergency incidents or error handling requirements).
- Fault**:** Missing performance requirements and other non-functional requirements have been omitted.

D) Analysis process

- Mistakes or misunderstandings in analysis of technical, operational, and financial feasibility/risks of requirements. This mistake can lead to the infeasible user needs, objectives, and other requirement among various faults.
- Mistakes in developing system models/ scenarios for analyzing requirements.
- Mistakes or lapses or misunderstanding while cutting the system into manageable pieces for analysis. This can lead to the omission of specific pieces or the redundancy of a particular piece.

- Misunderstandings or unresolved issues regarding complex system interfaces and unanticipated dependencies.
- Lack of any means for understanding and representing input and output space for all runs and in different circumstances.
- Inability to predict or guarantee the exact behavior for all kind of inputs and for different states or misunderstanding of desired behavior of the software.

Example

- Error: Mistakes (no exact behavior) during the analysis of requirements (e.g., in PGCS, there is no behavior specified when the driver takes a ticket and doesn't enter the parking space).
- Fault**:** Due to absence of the output specification, complete and clear mapping may not be specified.

E) Traceability Process

- Inadequate means of achieving traceability of requirements to predecessors and successors.
- Inadequate change management, including analysis of impacts and tracking changing requirements.

Example

- Error: Mistakes while establishing the traceability in requirements (e.g., in AAD system, the increase in the number of reserved tickets under some circumstances cannot be traced to any user requirement because it does not have any abstract predecessor or refined successor).

- Fault**:** Inability to track requirements causes extra requirements to be specified.

4.3.3 Documentation Errors

These kinds of errors occur due to the mistakes while organizing and specification of the requirements irrespective of whether the developers understood the requirements well.

A) Requirements Organization Errors

- Lapses or mistakes in listing or organizing requirements as a list of bullets.
- Ineffective selection of means of organization of requirements.
- Lack of awareness of logical way of organizing requirements.

Example

- Error: In PGCS, the specific requirements are listed without any logical organization.
- Fault**:** Requirements about synchronization of entry/exit gates is eliminated.

B) No Usage of a standard

- No usage of standard or means for documenting requirement specification.
- No usage of standard tool like a checklist of items to be included in requirement specification document.
- No usage of standard for notational descriptions (i.e., different notations or formatting used for same object in different sections is a part of misunderstanding).

Example

- Error: In the ATM system, no standard like checklist or IEEE standard is used and the scope of the system and performance requirements are omitted.
- Fault: Omitted and incomplete requirements (i.e., the scope of the system and performance requirements are missed out).

C) Specification Errors

- Mistakes or lapses while organizing the requirements irrespective of the adequacy of the organization method (e.g., the requirement intended for one set is mistakenly put into other set that do not bear any similarity).
- Omission of necessary attention checks at critical points can lead to deviation from intention.
- Using extraneous checks leads to repetition/omission of steps.
- Large time gap between solution formulation and its application.
- Misunderstandings in referencing to incorrect sections / requirements.
- Lack of awareness in description of the performance requirement specification and/or "implementation constraints".

Example

- Error: The requirement author had a means of organizing requirements, but he does mistakes while specification (e.g., in AAD system, the requirements regarding error recovery having clearly defined states is intended to be grouped into "mode of operation class" but is put into "user class").
- Fault**:** Mistakes in organizing can lead to ambiguity in requirements.

4.4 Family of Tools and Methods

This section describes the family of tools and methods that could be derived from the RET after it has been validated and, if necessary, updated through experimentation. This is future work, but is included to motivate the need for this type of a solution. Figure 7 shows the different tools that can assist developers at different points of software development stage. The framework of family of tools and methods is divided into two parts; the upper part describes the problem space and the lower describes the solution space. The problem space consists of three derivable methods with the high level goals of error prevention, error detection and error repairing. The solution space describes the technique used for each high level goal and their specific goals.

1. Prevention Techniques help developers avoid errors during development (prior to inspection) to reduce rework and their subsequent investments. To develop these techniques, I will use individual error information to describe *Preventive Steps* that can be followed in each phase to avoid that error. The preventive steps provide developers a way to learn from their mistakes and to communicate that knowledge among them. I will also narrow my focus by describing the preventive mechanisms for each class of errors in the classification.
2. Even after the prevention techniques are applied, some errors may not have been avoided. During the inspection process, an inspector can use the Detection Techniques to locate these errors. I will use the error classification scheme to determine the steps to include in reading techniques for detection. Developing a reading technique for each general type of error will narrow the inspector's focus and

the union of all techniques will provide comprehensive error detection. The error detection techniques will help increase the number of faults found, thereby improving software product quality.

3. After using the detection techniques during an inspection to find errors, those errors and the corresponding faults must be repaired. The Repairing Techniques will provide guidance to a developer that is specific to the type of error present. The errors should be repaired before moving on to the next phase.

This is a future task and is not covered in this thesis work.

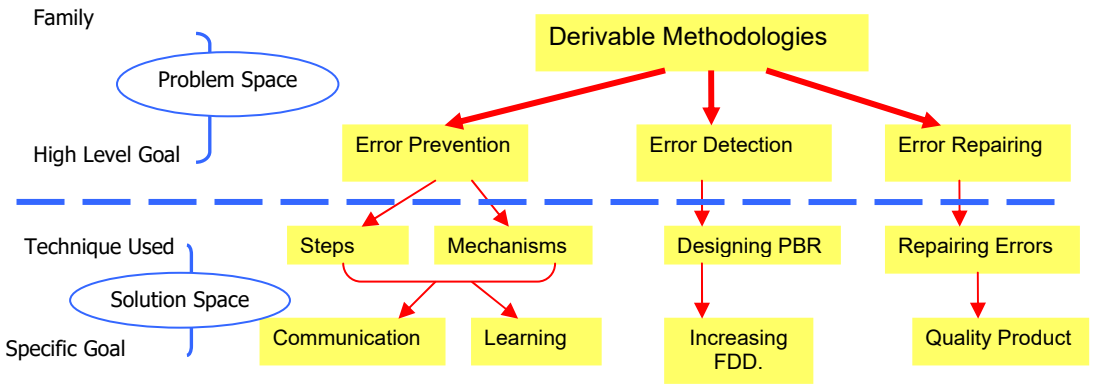


Figure 7: Framework of Family of Tools and Methods.

CHAPTER V

DESCRIPTION OF THE STUDY AND ANALYSIS

The goal of the experiment was to investigate the effectiveness of proposed solution in the requirements phase, by evaluating the usefulness of EAP and RET when analyzing a real software requirements specification. Primarily, we were interested in whether the EAP and RET improved productivity and did its intended task effectively along with the degree of insights provided.

The EAP used for developing error classification taxonomy is also evaluated. We have also investigated various variables that can affect the individual performance of developers using this QIA. We will use this information to improve the future experiment runs.

5.1 Experiment Design

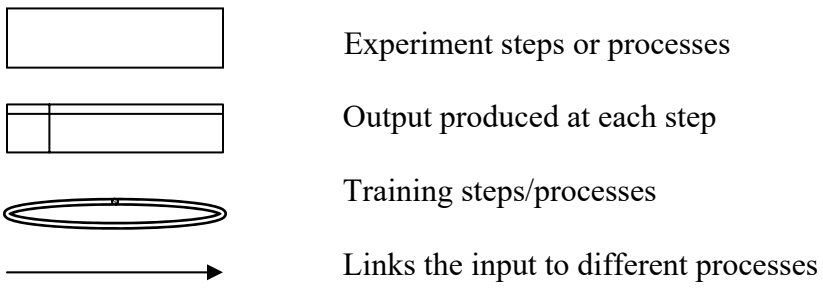
A controlled experiment was conducted for validation plan described in chapter 2. The experiment design consists of a methodology, including the setting and materials involved, followed by the experimental procedure that describes the experiment operation.

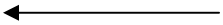
5.1.1 Methodology

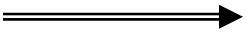
- Setting: The subjects of this study were sixteen (16) senior-level undergraduate students, majoring in either computer science or softwareengineering enrolled in the Software Engineering Senior Project course at Mississippi State University in the Fall 2005 semester. The subjects were divided into two teams of eight subjects each (by the course instructor, outside the scope of this study). Each team developed a different system.
- Materials: Each team developed a real system. They interacted with their customers to develop the software requirement specification for their system. Team 1 (T1) developed a system for managing ticket sales and seat assignments for the Starkville Community Theatre. Team 2 (T2) developed a system for managing apartments and town homes properties, including assignments of tenants, rent collection, and location of properties by potential renters.

5.1.2 Experiment Procedure

The study consisted of 5 steps and 2 training lectures. The remainder of this section describes each of those steps in detail. Figure 8 provides an overview of the experimental procedure, using the following notation:




Links different steps to outputs


Links processes together

1. *Step-1 Developing SRS*: In this step, each group interacted with its customer to develop a requirement specification document.
2. *Step-2 Inspecting SRS for faults*: When the SRS was complete, each student inspected it individually recording any faults detected. To conduct this inspection, each subject used a simple fault checklist. After all team members had conducted their individual inspections, they met together to consolidate their individual fault lists and agree on a team fault list.
3. *Training 1- Error Abstraction*: During this 40 minute session, the subjects were trained on the EAP and how to use it on their individual fault lists to abstract the underlying errors. The complete training description is provided in Appendix A.
4. *Step-3 Abstraction of Errors*: After the training, the subjects returned to their individual fault lists to abstract the underlying errors. These errors were documented in an Error-Fault List.
5. *Training 2- Requirement Error Classification*: This 120 minute session focused on the RET and its use. During the training, the RET was explained in detail. The students were then given a description of some fictitious systems (same as described in Section 4.3) along with a list of 12 errors to classify using the RET. The goal of this exercise was to ensure that the students understood the error classification process before using it on their own SRS documents. The students' classifications of these errors provide an idea of how well they understood the

classification scheme. To combat a potential validity threat of learning (i.e., they were better on error 10 than on error 1), we prepared two different lists (list A and list B) with the same set of errors, but in different orders and gave half of the subjects each list. The complete training description is provided in Appendix B.

6. *Step-4 Classification of Errors*: After the second training, the subjects returned to their individual error-fault lists from Step 3 to classify those errors into the RET. While doing this classification, the subjects were to record any additional errors they discovered as a result of using the RET.
7. *Step-5 Going back to locate more faults*: Finally, the students used the information in the RET about each error on their individual error-class lists to re-inspect the SRS to locate any additional faults that may be related to that error. Each student developed a new *error-fault list* (1 per student). Also each team developed its *team new error-fault list* (1 per team) and mapped it with their team *Error-Class list* (1 per team).
8. After completing all of these steps, each student completed a questionnaire to provide feedback regarding the EAP and RET. The complete training description is provided in Appendix C.

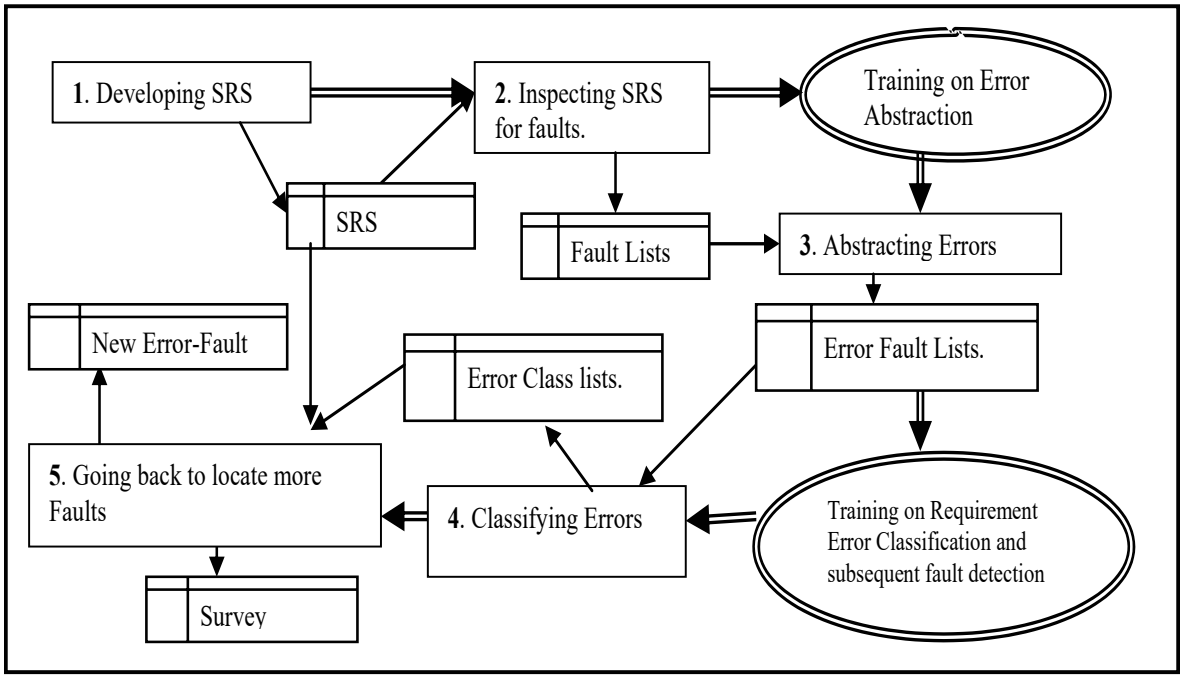


Figure 8: Description of Experimental Procedure

5.2 Data Collection Process

Data collection describes quantitative (Step 1-5) and qualitative (Survey and Questionnaire) data collected during the experiment procedure described in Section 5.1.2 and is used for analysis described in section 5.3. This section describes the individual data collected at different steps and maps it to the metrics described in Chapter 2.

5.2.1 Data Collected

A. SRS Inspection - Step 2:

   a. Each subject's fault-list (16).

   b. Group fault-list (1 per team).

B. Abstracting errors from faults- Step 3:

   a. Individual error-fault lists (16).

C. Training on RET and subsequent fault detection – Training 2:

   a. Error- Class lists on a practice run (16).

D. Classification of errors – Step 4:

   a. Individual error-class lists (16).

E. Re- Inspecting SRS – Step 5:

   a. Each subject's new error- fault list (16).

   b. Group new error-fault list (1 per team).

   c. Group error- class list (1 per team).

F. Survey and Questionnaire:

   a. Characterizing attributes of error abstraction process.

   b. Characterizing attributes of requirement error taxonomy.

   c. Characterizing usage of training procedures.

5.2.2 Mapping of Data and Metrics

Table 5.1: Mapping of Data and Metrics.

Metric	Data Collected
M1: Errors and Fault caused by each error class.	A (a, b), B(c), D (a), E (a, b, c).
M2: Contribution of human reasons	A (a, b), B(c), D (a), E (a, b, c).
M3: Density of unclassified errors.	D (a)
M4: Characterizing RET	F (a)
M5: Feedback on RET and EAP	F (a, b, c)
M6: Causes of problem areas	A (a, b), B(c), D (a), E (a, b, c).
M7: Impact of QIA on productivity	A (a, b), B(c), D (a), E (a, b, c).

5.3. Data Analysis

I will describe the analysis of data pertaining to each hypothesis described in Chapter 2. Also, I will describe the analysis of data relative to the research questions from Chapter 2. Chapter 6 provides an interpretation of results in light of the overall study goals. I have used an alpha value of 0.05 for judging significance of my results.

5.3.1 Fault Detection Effectiveness and Efficiency

It was first important to determine whether using our QIA (EAP and RET) provided any increase in effectiveness or efficiency over the standard fault checklist process. To address effectiveness, I calculated the number of faults detected during the first inspection (prior to learning about the EAP) to the number of faults detected during the re-inspection (after learning EAP), Figure 9 compares the number of faults detected during Step-2 of experiment process (before EAP) and the total faults found (including the faults found after EAP). These results show that there was a 75% and 154% increase in total fault detection provided by our QIA for Team 1 and Team 2 respectively. The percentage increase in the effectiveness for Team 2 is almost double than the increase for Team 1. One likely reason for this could be that Team 2 was developing software from less common domain (managing rental properties), and so they made more mistakes than Team 1 who worked on more common system (event ticket sales).

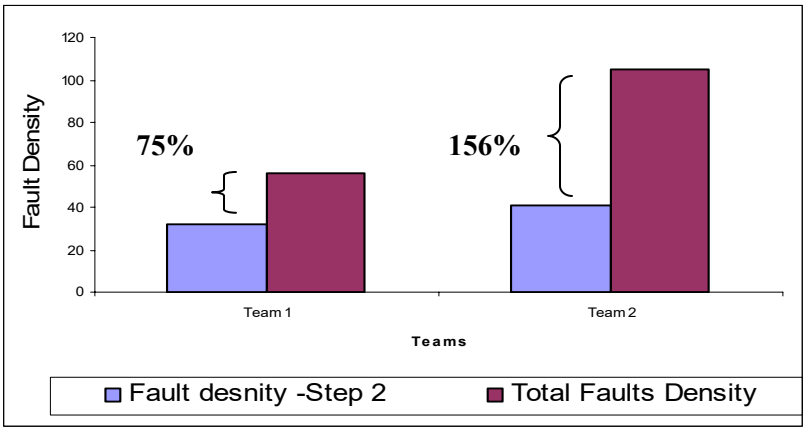


Figure 9: Comparison of Effectiveness of Both Teams.

To further investigate the large increase in the team fault detection, I examined the individual subjects to determine if that increase was consistent throughout the sample or concentrated in only a few subjects. Figure 10 shows the percentage increase seen by each subject from Team 1, and Figure 11 shows the percentage increase seen by each subject from Team 2. All subjects in both teams displayed some increase when using the QIA, while those on T2 seem to be more evenly distributed.

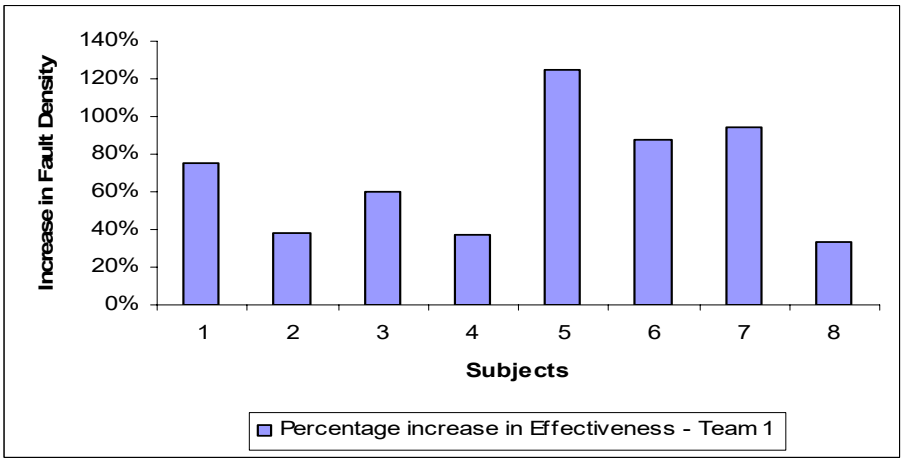


Figure 10: Percentage Increase in Effectiveness for 8 Subjects: Team 1.

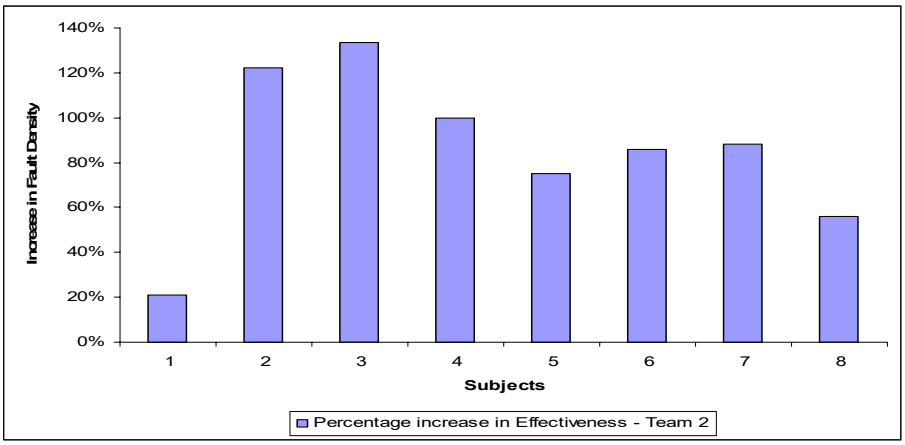


Figure 11: Percentage Increase in Effectiveness for 8 Subjects: Team 2

.

I ran a one-sample t-test separately for sample of all the subjects in each team. In this analysis, I tested whether the number of faults found during the re-inspection was significantly different from zero (0). This is because if the RET was not helpful, we would not have expected any new faults to be found. The results of this test show that the fault detection for both teams is significantly higher than zero (p= 0.02 [T1]; p= 0.00 [T2]).

To address efficiency, I analyzed the impact QIA had on the fault detection rates of T1 and T2 as shown in Figure 12. Because of the multiple steps involved in the process, it was not clear how to arrive at an effort figure to use for comparison. Therefore, Figure 12 compares 3 fault rates for each team. The value for A is computed by dividing the number of faults found during the pre-EAP inspection by the number of hours that the inspection took (Step 2). The value for B is computed by dividing the number of faults found during re-inspection after using EAP (Step 5) divided by the time taken only for the re-inspection (Step 5). The value for C is computed by dividing the number of faults

found during the re-inspection after using EAP (Step 5) divided by the total time for error abstraction, error classification, and re-inspection (Steps 3, 4 and 5). C is probably the most accurate figure because it considers the time spent in using the whole QIA (i.e., both EAP and RET).

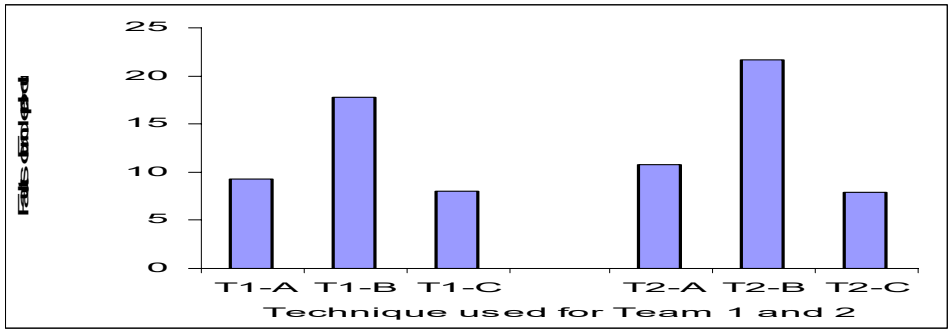


Figure 12: Comparison of Efficiency of Both Teams.

As Figure 12 shows, there was a large increase in detection rate when only considering the effort for re-inspection. Conversely, when taking into account all of the effort associated with the QIA (i.e, EAP +RET), the fault detection rate is slightly lower than the fault detection rate using the standard fault checklist method. However, this small decrease is offset by the benefit of the additional faults detected and is therefore a worthwhile use of effort.

I also used a one-sample t-test for comparing the reduction in fault rates when using QIA is not significant for either team.

5.3.2 Usefulness of RET

I evaluated the RET using feedback from eight essential attributes: simplicity (sim), understandability (under), applicability (usa), intuitiveness (intui), orthogonality (orth),

comprehensiveness (compr), usefulness (use) and uniformity across products (unifor) as shown in Figure 13. For each attribute, each subject was asked to rate the RET as 1-Low, 2-Medium, or 3-High.

Figure 13 shows the distribution of those rankings for the 16 subjects. The results in the figure show that in general the RET was viewed favorably for all attributes. There are no cases where an attribute received more low ratings than high ratings.

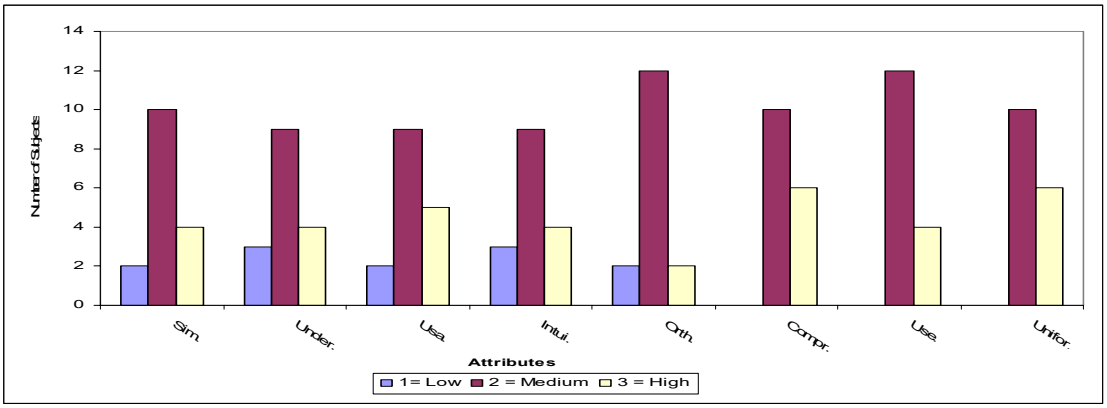


Figure 13: Frequency Distribution of Weights.

Furthermore, I asked the subjects to report on the adequacy of the error classes for the given task and ease of placing errors in the appropriate error class. Figure 14 summarizes the responses of the subjects using the same scale as in Figure 13. Only one (1) subject gave the adequacy of error class descriptions a low weight.

Figure 14: Adequacy of Error Classes.

In addition to the subject's qualitative evaluation, I analyzed the error-class lists to determine if there were any errors that could not be classified in the RET. I found all the errors were classified. The qualitative and quantitative data together provide sufficient evidence to believe that the RET was complete and easy to use.

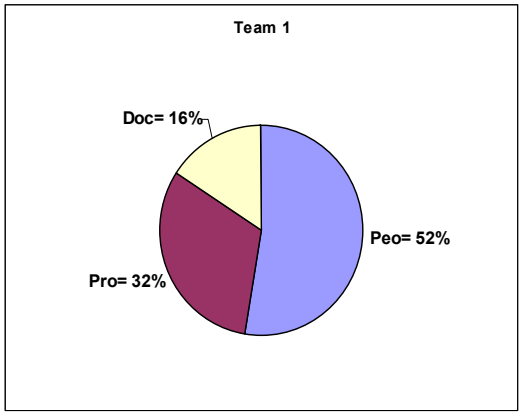
5.3.3 Insights Provided

I analyzed 3 high-level error types and 14 detailed error classes (Figure 6) to understand their contribution towards improving the quality of software. I wanted to know if errors and faults were evenly distributed among the error types and error classes, and also whether any particular type was a major cause of redundant, time consuming or multiple faults. For these analyses, I examined the errors and the resulting faults separately to determine if the effects of the error types and classes were any different.

5.3.3.1 Error Types vs. Error and Fault Density

Figure 15 and Figure 16 compares the percentage contribution of the three high-level error types, People (Peo), Process (Pro) and Documentation (Doc) errors to total error

density for Team 1 (T1) and Team 2 (T2) respectively. These percentages were computed using each team's error class list to count the number of errors of each type. Graphically the distribution shows that people error contributed to most of the errors for both teams. However, using an ANOVA test did not show any significant difference. Similarly, analysis of the contributions of error types to fault injection density for the Team 1 and Team 2 is shown in Figure 17 and Figure 18 respectively. To perform this analysis, we used each team's error-class lists and error- fault list from Step 3 and error-fault list from Step 5 to account for all the faults found.



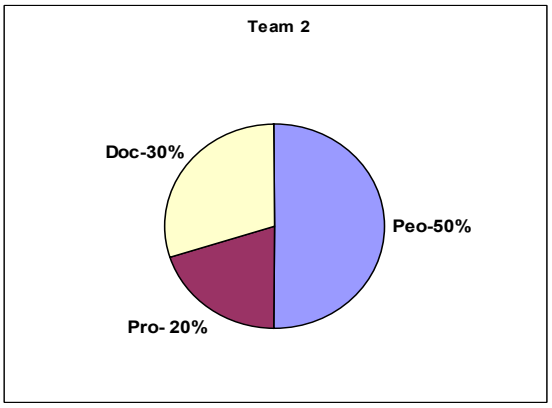
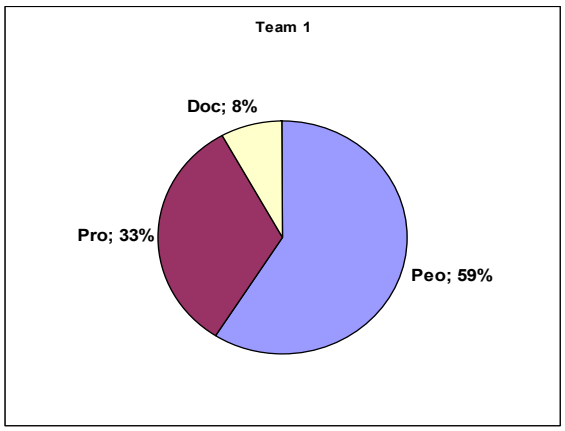


Figure 15: Error Types vs. Error Density: T1 Figure 16: Error Types vs Error Density: T2

Again the distributions in Figure 17 and 18 reveal as expected, that people errors causes the highest number of faults for both teams. However, the results of the ANOVA support the intuition that similar to the results for error density, fault density is also evenly distributed over the three error types.



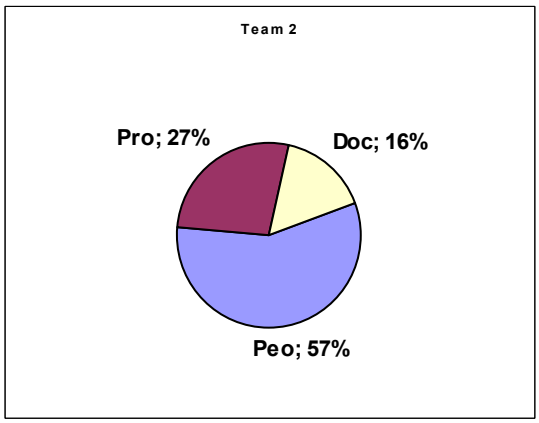


Figure 17: Error Type vs. Fault Density: T1 Figure 18: Error Type vs. Fault Density: T2

5.3.3.2 Error Classes vs. Error and Fault Density

In addition to examining the contribution of three higher level error types, I also wanted to examine the contribution of 14 more detailed error classes. Figure 19 shows the percent of errors contributed by each of 14 error classes (grouped by their high-level error type). Results of an ANOVA test shows that while the contribution of error density among the 14 error classes varies, it is not significantly different. Figure 19 show that there was at least one error from each error class in our taxonomy indicating that all of our error classes are necessary. Therefore I can conclude that not only is error density evenly distributed over the high-level error types, it is also evenly distributed over the 14 detailed error classes. Similarly for fault detection, Figure 20 shows the percent of faults that were caused by each of the 14 error classes. The results of the ANOVA test again show that there is no significant difference among the contribution of the 14 error types to the overall fault density. Similar to errors, Figure 20 shows that there is at least one

fault caused by an error from each class. This result gives more evidence to the validity of the error classes.

5.3.3.3 Major Causes of Redundant, Multiple and Time Consuming Faults

I wanted to determine if any of the three high-level error types was responsible for a large number of redundant faults (i.e., faults that appeared on more than one team member's fault list during Step 2 and Step 5). This information helps determine where improvement could be made to reduce the overlap. Figure 21 compares the contribution of each error type for both the teams. The data shows that people errors have a higher contribution than other error types for both teams. This result was statistically significant using an ANOVA ($F_{2,5}$ = 9.700, p = 0.049). An additional post-hoc test (the Tukey test) showed that the mean difference between People and Documentation errors is significant (p = 0.048).

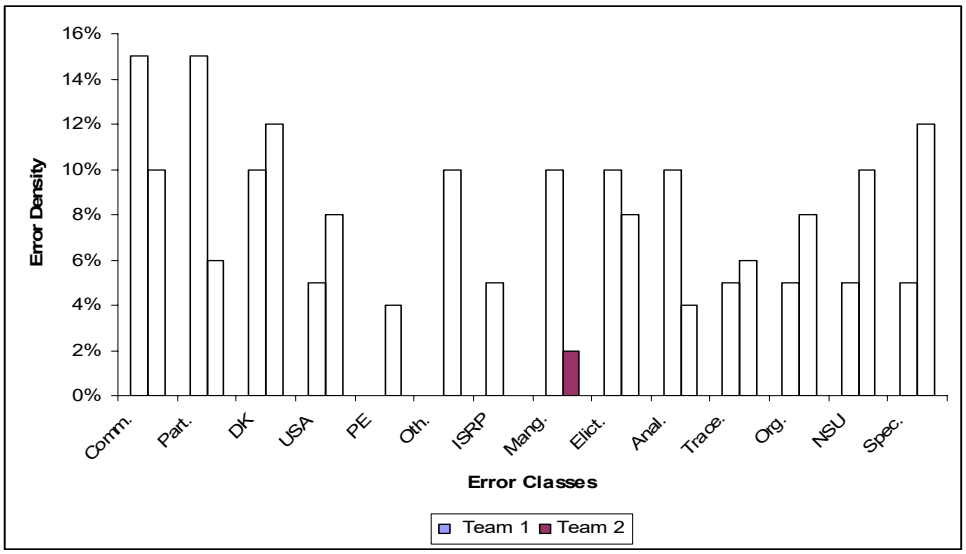


Figure 19: Contribution of Error Class to Error Density

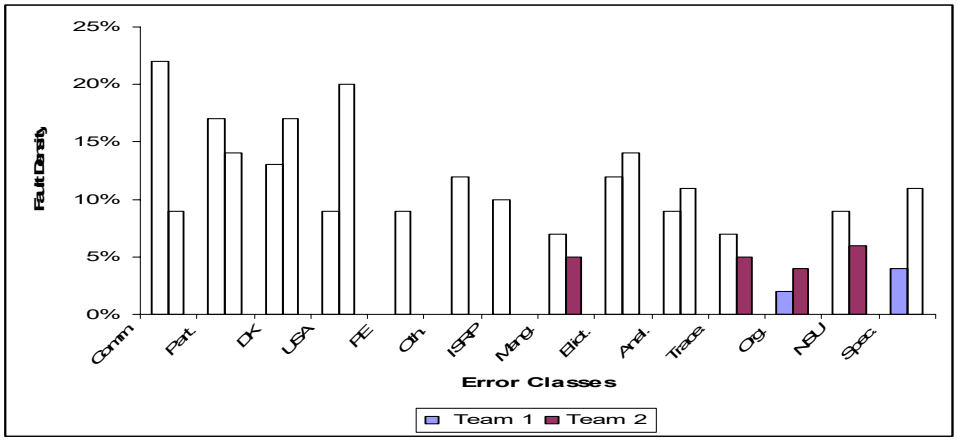


Figure 20: Contribution of Error Class to Fault Density

Figure 22 shows the contribution of error types to most time consuming faults for both teams. I used the Team "Fault Lists" produced during Step 2 (Figure 2) to record high time consuming faults. Also, I used the consolidated "Error-Fault" lists for each team produced at Step 5 (Figure 8) to find more time consuming faults. We used this list of time consuming faults to analyze their major cause for both teams. The base value for time consuming faults was taken to be "greater than or equal to 15 minutes" which was greater than the average time taken to find most of other faults. The Process errors have a higher contribution than other error types for both teams.

Results from the ANOVA test show that $F_{2,5} = 9.500$, $p = 0.05$. Thus, the difference between mean contributions of error types is significant. Also, the Tukey test shows that the mean difference between Process and Documentation error type is significant with significance value of 0.046.

Another important aspect of the errors is whether an error is responsible for multiple faults. We wanted to determine if errors from any of the three high-level error types were

more likely to result in multiple faults. To perform this analysis, we examined the team fault lists and the team error-fault lists, produced in Steps 2 and 5 respectively.

Figure 23 shows the results of this analysis. The results showed that People errors were more likely than the other two types to cause multiple faults. This difference is not significant with our alpha value of .05 when evaluated with an ANOVA ($F_{2,5} = 8.167$, $p = .061$).

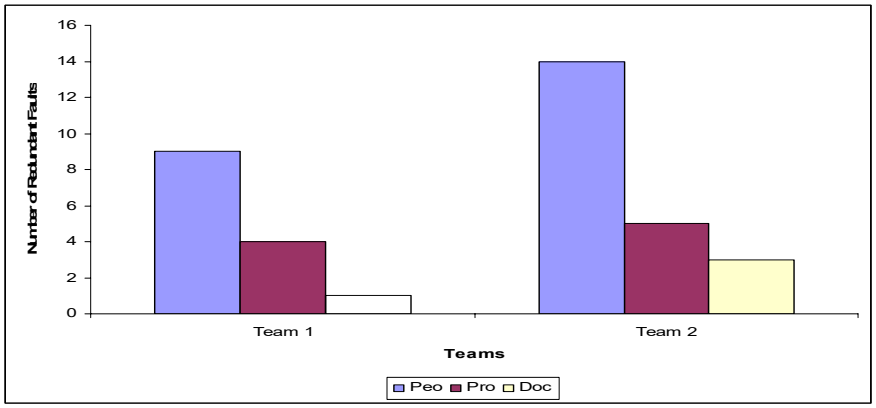


Figure 21: Comparing Causes of Redundant Faults

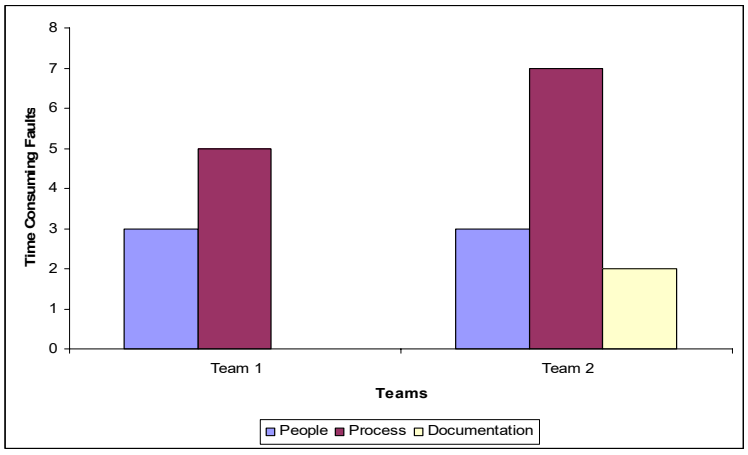


Figure 22: Comparing Causes of Time Consuming Faults

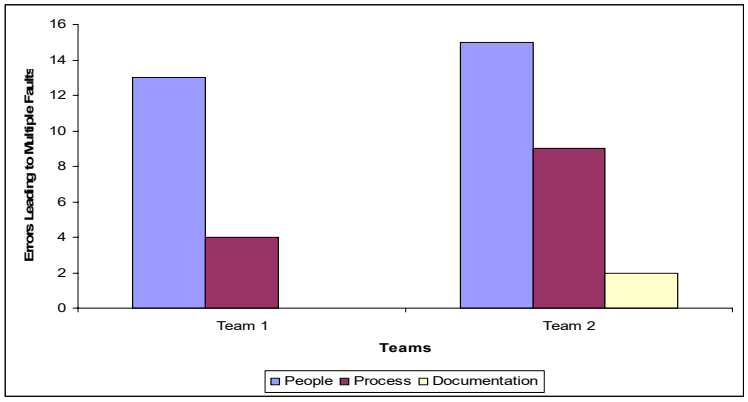


Figure 23: Comparing Causes of Multiple Faults

5.3.4 Contribution of Human Cognition to Fault Density

One of the contributions of our RET was the use of research from human cognition to augment the error classes developed from software engineering resources alone. I analyzed the percentage of errors from each team that were classified into detailed classes that were created based on our human cognition research. The results, shown in Figure 24, indicate that a meaningful contribution was made by human cognition errors for both teams (20% and 25%). This result indicates that the use of research from these fields was effective in bolstering the RET.

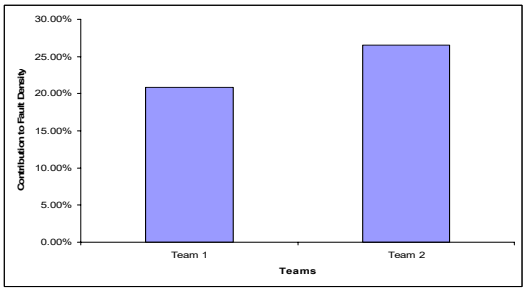


Figure 24: Contribution for Each Team

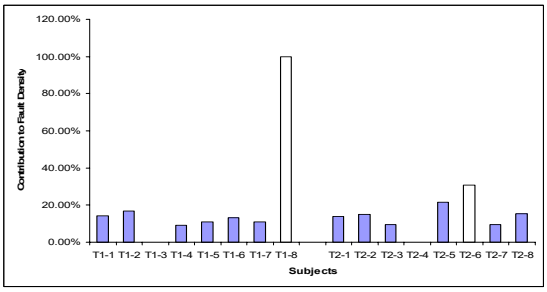


Figure 25: Contribution for Each Subject

Furthermore, Figure 25 breaks this result down by individual subject to show that in most cases each subject found errors that were related to human cognition, with one subject finding only human cognition-related errors.

5.3.5 Effects of Other Variables

Finally, I analyzed the effects of different independent variables on the productivity of subjects. The independent variables analyzed were (as following):

1. Degree of Process Conformance.
2. Performance on Practice Run.
3. Usefulness of Training Procedure.
4. Effort Applied.

5.3.5.1 Process Conformance vs. Fault Detection Density

I analyzed the effect of degree of process conformance on individual fault density. I used the qualitative data on process conformances at three different points during the experiment (Step 3, 4 and 5 in Figure 2) and the median value of these three process conformances. Results show that the increase in fault density only depends on the process conformance during error abstraction process (Step 3) and median process conformance with the significance values of 0.004 and 0.030 respectively.

5.3.5.2 Effort vs. Fault Detection Rate

Figure 26 shows the relationship between effort expended during the EAP and individual fault detection rate (i.e, faults found per hour). The effort value was computed using the sum of the time spent in hours during the three parts of the experiment

procedure (abstraction, classification, and re-inspection) and plotted against fault rate for each subject. The trend line indicates that there is a significant positive relationship between effort and fault rate (p = .02). So, not only does more effort result in more faults (an expected result); it also results in more faults per hour (an unexpected and very positive result).

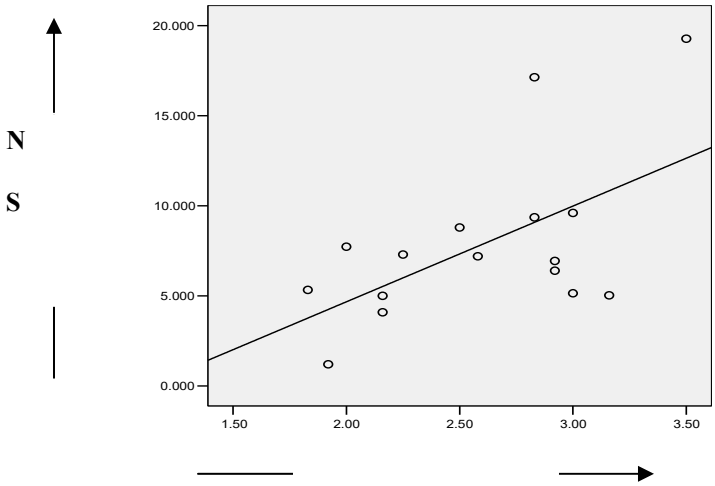


Figure 26: Effort vs. Fault Detection Rate

5.3.5.3 Usefulness of Training vs. Fault Detection Density

We conducted two training sessions with the students and both sessions focused on different aspects, one focusing on abstracting errors from faults (Training 1) and the other focusing on RET (Training 2). In order to understand the relationship between how useful the students viewed the training and their subsequent fault detection density, we compared each training session separately.

The analysis showed a significant relationship for Training session 1 (the EAP) with p = .04, but not for Training Session 2 (the RET). This result indicates that understanding the process of error abstraction was more important than understanding the classification scheme. This result is shown graphically in Figure 27.

5.3.5.4 Performance on Practice Run vs. Fault Detection Density

Finally, we compared the students' performance on the classroom exercise of classifying example errors during Training session 2 against their subsequent fault detection density. The goal with this analysis was to understand whether performance during a practice run could be an accurate predictor of performance on a real project.

To perform this analysis, we counted the number of errors the student correctly classified out of the 12 example errors and compared it against their fault detection density. Figure 28 shows that there was a significant positive relationship between these two variables (p = .024). This result indicates that the practice run is a good predictor of performance.

5.3.6 Analysis of Qualitative Data

In addition to the quantitative data, we also collected some qualitative feedback from students regarding the usefulness of the error abstraction process. Figure 29 shows the students rating of the EAP on five different attributes: CIU - confidence in using error abstraction process, MOR - meaningfulness of results, HIURP - usage in understanding real problems, WOES - worthiness of effort spent, and CERM - confidence of errors

being real misunderstandings. For these attributes, the students had moderate to high opinion of the EAP.

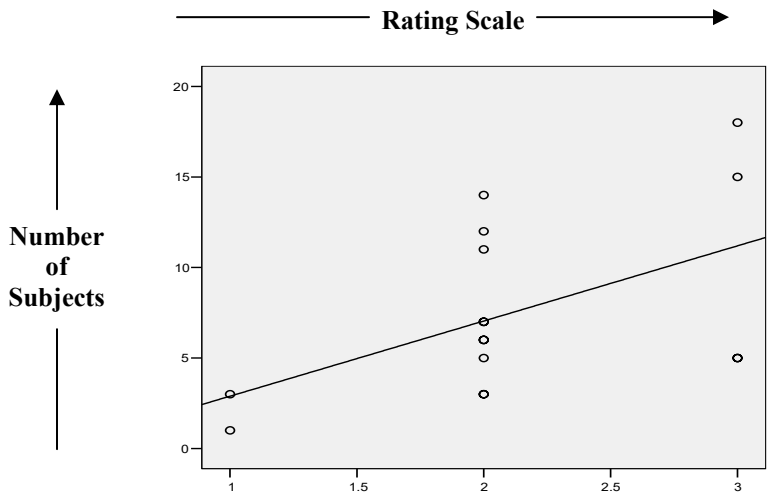


Figure 27: Training 1 vs. Fault Density

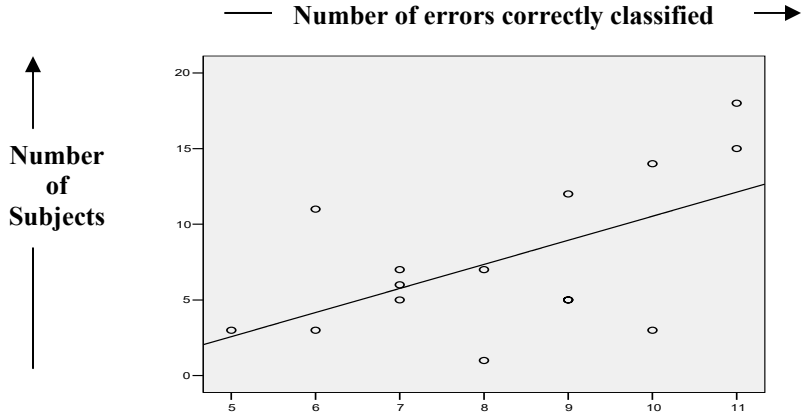


Figure 28: Performance vs. Fault Density

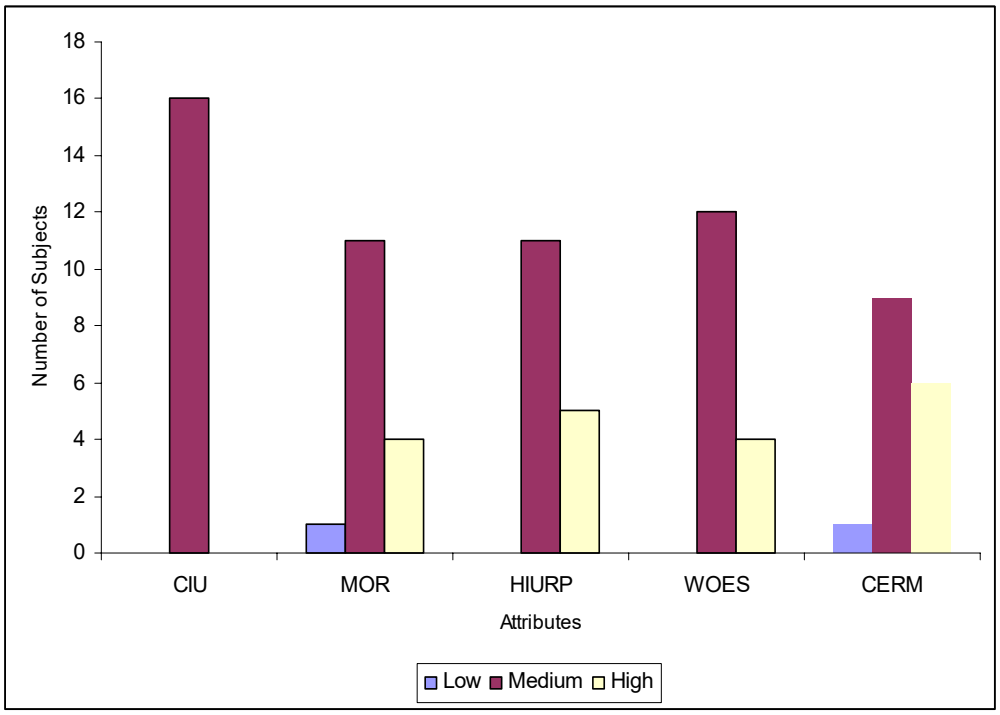


Figure 29: Frequency Distribution for EAP

CHAPTER VI

DISCUSSION OF RESULTS

This chapter presents the discussion of the results and evaluates hypotheses by answering the research questions in Chapter 2. Section 6.1 discusses results in relevance to research questions from Chapter 2. Section 6.3 discusses the limitations of this study, describing the threats to validity.

6.1 Research Questions

1. Does the RET satisfies all essential attributes (i.e, simplicity, understandability, applicability, intuitiveness, orthogonality, comprehensiveness, usefulness and uniformity across products) [33]? What is the degree of adequacy of error types and error classes in RET?

Analysis of the characterization of RET on eight essential attributes is described in form of a frequency distribution histogram in Figure 13. The result show that the RET satisfies all the attributes, with some of the attributes (usability, usefulness, comprehensive, uniformity) better than others. Using this feedback, the RET will be updated and refined for further experimentation and validation.

Figure 14 summarizes the responses of the subjects on adequacy of error class and placement of each error. The result shows that the error classification was

adequate. Also, an analysis of error-class list does not show any error that could not be classified in any error class.

2. Does the QIA improve productivity of developers?

In terms of effectiveness, results from analysis in Section 5.3.1 allow us to say that the QIA improves the effectiveness of both the teams by a large margin (75% and 156%). Also statistical results show that there is increase in fault detection density during re inspection.

Furthermore, there is a healthy increase in the effectiveness for each of the 16 subjects. In addition, there was a large increase in efficiency when only the inspection effort is considered. There was a small decrease in efficiency when the full EAP effort is considered; however, this decrease is not statistically different.

Combining the effectiveness and efficiency results allows us to conclude that the QIA was both effective and efficient and should be studied further to continue to improve it.

3. Does the QIA provide useful insights into the problems in the requirement phase of the development process?

The results from Section 5.3.3 showed that while people errors made the largest contribution to error and fault injection density, statistically the errors and faults were evenly distributed across the classes. Even so, analysis showed that people errors contributed significantly more redundant faults than the other two classes. Similarly process errors contributed significantly more time consuming faults than the other two classes. Finally, people errors contribute significantly more errors causing multiple faults.

These results provide insights into the major problem causing error types; and provide useful information to developers. These results give the confidence that the error types in the RET are valid and provide a good coverage of the overall requirements error space.

4. What factors affect the performance of the developers during the application of QIA on requirements document?

The results from Section 5.3.5 showed that process conformance during QIA, median process conformance and performance on the practice run all had a significant impact on the fault detection density. In addition, the defect detection density was significantly correlated with the perceived usefulness of the EAP training but not with the RET training. Surprisingly, effort spent during the QIA had a significant effect on the fault detection rate.

These results allow drawing the following conclusions:

- To increase fault detection density subjects must follow the process during error abstraction.
- An increase in effort spent is likely to lead to an increase in fault detection rate.
- A subject's performance on a practice run can be used to predict their fault detection density using the QIA.
- Useful training on how to abstract errors from faults is necessary for improving fault detection effectiveness.

5. Are all the errors and their resulting faults in each error class detectable or feasible?

The results show that the technique (i.e., RET) developed by application of EAP is a potential solution. Figures 19 and 20 shows that at least there was one error from each class that was detected and also caused fault. The EAP has provided a good solution that provides comprehensive list of errors.

6. What is the contribution of the research from human cognition, psychology and similar fields to help locate more errors and corresponding faults?

The results from Section 5.3.4 indicated that between 20% and 25% of the errors found could be classified as human cognition errors. This result gives us confidence that using research from other related fields like human cognition is beneficial for the RET. Therefore, using research from these fields is an important component of an EAP to develop sound error taxonomy.

7. Does mapping back to human errors improve error taxonomy?

Mapping back to human errors that were surveyed before developing error taxonomy helped as some of the errors were redundant (as they were similar) and missing (overlooked). It removed redundancy making it simpler and comprehensive.

6.2 Limitations of this Study

In this section we discuss both the threats to validity that we were able to address, and those that we were unable to address. In order to avoid a learning effect during the classroom practice session, the order of the items being classified was randomized. Also, to reduce the external validity threat that would be caused by using a toy example, the students were working on real systems and interacted with real clients to develop the requirements specification document.

However, there were also some threats to validity that were not addressed by the experimental design. While the students did work on a real system, the study was still done in a classroom environment affecting the external validity of the study. Another important threat is that we did not have a control group. During the re-inspection where the students use the EAP, we did not have a group of subjects doing a re-inspection using their standard technique to compare against. Therefore, it is possible that a portion of the effect we observed may have been caused by the fact that the students were inspecting the artifact for a second time.

CHAPTER VII

CONCLUSION

This chapter revisits the hypotheses formed in Chapter 2 and evaluates them in Section 7.1 using the answer of the research questions discussed in Section 6.1. This Chapter also talks about the contributions of this work and motivations for future work in Sections 7.2 and 7.3 followed by publication plan.

7.1 Evaluation of Hypotheses

Hypothesis 1: EAP is a feasible way of creating and improving effective quality improvement approach (QIA) than the approaches based on faults and other similar approaches to quality improvement.

From the answers of the research questions 5, 6 and 7, it can be said that the EAP is an effective process of developing technique that can improve software quality. The individual elements of EAP have been evaluated, and they contribute to the overall solution.

Hypothesis 2: This QIA based on error information improves productivity of developers and software quality than the approaches based on fault classifications and similar techniques.

From our analysis in this research, we can say that our results are good evidence that support this hypothesis. The QIA based on error information improved productivity of developers significantly and performed better when compared with the technique based on fault information. To validate it, we plan to run more experiments in the future and compare this technique with other fault detection techniques to extrapolate our results.

Hypothesis 3: The RET fulfills criterion [33], provides useful insights and is useful for developers.

From the analysis and evaluation of research questions 1 and 3, it is right to say that the RET fulfills the criterion [33], provided some useful insights into the requirement phase, and proved to be useful for software developers.

7.2 Contribution to Research and Practice Communities

The contributions of my research include providing the research community a new perspective for investigation into means of ensuring software quality. This is an initial investigation, so the results from this study will motivate me and other interested researchers to investigate the effectiveness of the tools and methods developed using this technique. Also, the application of the error abstraction process to the subsequent phases of the software lifecycle will help the research community gain confidence in the usage of the error abstraction throughout the development process. This confidence will help the research and practice communities use tools and methods at various points during the development process to track the progress of quality product, as it is developed and to predict and measure software quality.

Also, the validation of the error abstraction process throughout the software process will help in developing a complete verification and validation process that can change the present condition of software quality in the near future.

7.3 Motivation for Future Research

Based on the results of the study, we conclude that both the EAP and RET provide a benefit to developers who use them. The feedback provided by the students will allow us to refine the EAP and improve RET to make them better for future studies.

Our future plans include building fault detection techniques based on the RET and creating a Design Error Classification Taxonomy using the same approach. To get an initial idea of the value of these two ideas, we asked the students for feedback after the study was over. We asked the students whether they thought that it would be worthwhile to develop error taxonomy for later lifecycle phases (ECTSP) and to derive tools and methods for improving software quality based on the RET (DTRE). The responses of the students are shown in Figure 30. Based on the positive results of the study and these responses, we believe that it will be a worthwhile endeavor to pursue these ideas.

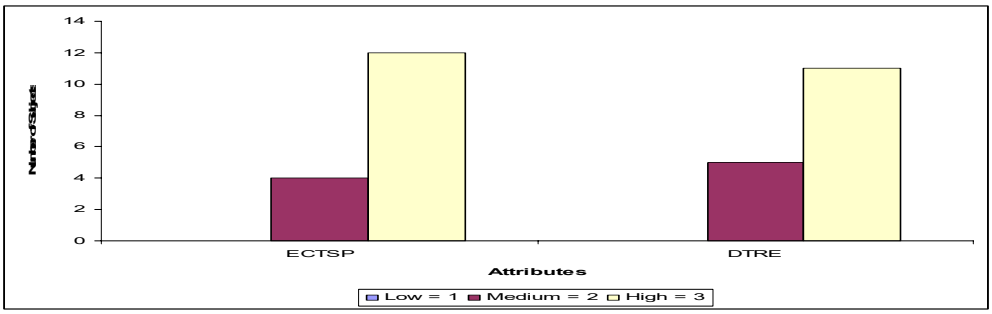


Figure 30: Frequency Distribution for Motivation

In the future, we plan to continue to conduct experiments to empirically validate the EAP and continue to refine both the EAP and RET with feedback from subjects. Collecting and analyzing additional data will allow us to gain more useful insights into improving the quality of software using the EAP and RET at the requirements stages.

7.4 Publication Plan

My publication plan consists of writing three papers from this research. The first paper will include a description of literature survey, description of our proposed framework design, and the "Requirement Error Classification Taxonomy". I plan to send this paper to ACM Surveys or the Journal of Information and Software Technology in the month of May. After the analysis was complete, I wrote a paper describing the study and the impact of the results on software quality. I submitted this paper to the International Symposium on Empirical Software Engineering. I will combine the description of my approach and the important results into a comprehensive paper. I will submit that paper to the Journal of System and Software or IEEE Transactions in Software Engineering. I plan to complete this paper during the summer.

REFERENCES

[1] A. F. Ackerman, L. S. Buchwalk, and F. H. Lewski, "Software Inspections: An Effective Verification Process," *IEEE Software,* May 1989, pp. 31-36.

[2] M. Alexander, "Towards a System Approach to Human Error Investigation," *Proceedings: 48th Human Factors and Ergonomics Society Meeting*, 2004, pp. 2436-2440.

[3] S. Alistair, G. Julia, M. Shailey, "Human Errors and System Requirements," *Proceedings: 4th IEEE International Symposium on Requirements Engineering*, 1999, p.23.

[4] S. G. Alistair, M. Neil, M. Shailey, M. Darrel, "Supporting Scenario Based Requirements Engineering," *IEEE Transactions on Software Engineering*, vol. 24, no.12, Dec. 1998, pp. 1072-1088.

[5] V. Basili, "Evolving and Packaging Reading Technologies," *Journal of Systems and Software*, vol. 38, no. 1, July 1997, pp.3-12.

[6] V. R. Basili, S. Green, O. Laitenberger, F. Lanubile, F. Shull, S. Sorumgard, and M. Zelkowitz, "The Empirical Investigation of Perspective Based Reading," *Empirical Software Engineering: An International Journal*, vol. 1, no. 2, 1996, pp. 133-164.

[7] V. R. Basili, F. Shull, I. Rus, and O. Laitenberger, "Improving Software Inspections by Using Reading Techniques", *Proceedings: 23rd International Conference on Software Engineering* (ICSE), June 2001, pp. 726-727.

[8] M. R. Blackburn, R. Busser, A. Nauman, *Eliminating Requirement Defects and Automating Test*, Software Productivity Consortium NFP, 2001; http://www.software.org/pub/externalpapers/tcs_2001.pdf.

[9] B. Boehm, "Validating and Verifying Software Requirements," *IEEE Software*, vol.1, no. 1, 1984, pp. 75-88.

[10] B. Boehm and V. R. Basili, "Software Defect Reduction Top 10 List," *Computer*, vol. 34, no. 1, Jan. 2001, pp. 135-137.

[11] J. V. Bukowski, W. M. Goble, "Practical Lessons for Improving Software Quality," *Proceedings: 13th Reliability and Maintainability Symposium*, IEEE Computer Society, Jan. 1990, pp. 436-440.

[12] G. Caldiera, F. Shull and F. Lanubile "Studies on Reading Techniques", *Proceedings: 21st Annual Software Engineering Workshop,* Software Engineering Laboratory Series (SEL-96002), Greenbelt, Maryland, Dec. 1996, pp. 59-65.

[13] N. D. Card, "Learning from Our Mistakes with Defect Causal Analysis," *IEEE Software,* vol. 15, no. 1, Jan. 1998, pp. 56-63.

[14] J. Carver, "Impact of Background and Experience on Software Inspections," doctoral dissertation, Department of Computer Science, University of Maryland, College Park, Maryland, April 2003; http://www.cse.msstate.edu/~carver/Papers/Dissertation.pdf.

[15] J. Chaar, M. Halliday, I. Bhandari, and R. Chillarege, "In-Process Evaluation for Software Inspection and Test," *IEEE Transactions on Software Engineering*, vol. 19, no. 11, Nov. 1993, pp.1055-1070.

[16] R. Chillarege, I. Bhandari, J. Chaar, M. Halliday, D. Moebus, B. Ray, and M. Y. Wong, "Orthogonal Defect Classification - A Concept for In-Process Measurements," *IEEE Transactions on Software Engineering*, vol. 18, no. 11, Nov. 1992, pp. 943-956.

[17] *Computer Aided Dispatch System for the London Ambulance Service: Software Requirement Specification*, tech. report MSU-030429 Version 0.4, Computer Science Dept., Mississippi State Univ., Jan. 2003.

[18] W. J. Cristopher, "The Cost of Errors in Software Development: Evidence from Industry," *The Journal of System and Software*, vol. 62, no. 1, May 2002, pp. 1-9.

[19] G. Damele, G. Bazzana, F. Andreis, S. Aquilio, "Process Improvement through Root Cause Analysis," *Proceedings: 3rd International Conference on Achieving Quality in Software*, vol.12, no.2, 1996, pp. 35-47.

[20] E. David, "Understanding Human Behavior and Error", *Human Reliability Associates Ltd.*, vol. 3, no. 1, Oct. 1979, pp. 31-37.

[21] W. A. Douglas, A.S. Scott, "Applying the Human Factors Analysis and Classification System to the Analysis of Commercial Aviation Accident Data," *Proceedings: 11th International Symposium on Aviation Psychology*, 2001, pp. 59-86.

[22] M. E. Fagan, "Design and Code Inspections to Reduce Errors in Program Development," *IBM Systems Journal*, vol.15, no. 3, 1976, pp. 182- 211.

[23] M. S. Feather, "Infusing and selecting V&V Activities," *Proceedings: V&V State of the Art Foundation, Laurel*, Maryland, Oct. 2002; http://www.cs.clemson.edu/~found04/Foundations02/Session_Papers/A5T5_feather.pdf.

[24] W. Florac, *Software Quality Measurement: A Framework for Counting Problems and Defect,* tech. report CMU/SEI-92-TR-22, Software Engineering Institute, Sept. 1992; http://www.sei.cmu.edu/pub/documents/92.reports/pdf/tr22.92.pdf.

[25] D. A. Gaitros, "Common Errors in Large Software Development Projects," *The Journal of Defense Software Engineering*, vol. 12, no. 6, Mar. 2004, pp. 21-25.

[26] J. A. Goguen, C. Linde, "Techniques for Requirements Elicitations," *Proceedings: International Conference on Software Engineering*, vol. 12, no.3, 1993, pp. 152-164.

[27] J. Holt, T. Newman, G. Mathieson, "A Realistic Simulation of Decision Making Behavior," *Journal of the Experiment Analysis of Behavior*, vol. 69, no. 3, May 1998, pp. 355-364.

[28] IEEE Std. 1044-1993, *IEEE Standard Classification for Software Anomalies*, IEEE Computer Society, New york, 1993.

[29] M. S. Jaffe, G. L. Nancy, M. P. E. Heimdahl, B. E. Melhart, "Software Requirement Analysis for Real Time Process-Control Systems," *IEEE Transactions on Software Engineering*, vol.17, no.3, Mar. 1991, pp. 241-257.

[30] T. M. Khoshgoftaar, J. C. Munson, "Predicting Software Development Errors Using Software Complexity Metrics," *IEEE Journal on Selected Areas in Communication*, vol.8, no.2, Feb. 1990, pp. 253-261.

[31] F. Lanubile, F. Shull, V.R. Basili, "Experimenting with Error Abstraction in Requirements Documents," *Proceedings: 5th International Symposium on Software Metrics* (METRICS'98), Nov. 1998, pp. 114-121.

[32] C. P. Lawrence, I. Kosuke, "Design Process Error-Proofing: Failure Modes and Effects Analysis of the Design Process," *Proceedings: Design Engineering Technical Conference* (DETC '03), Illinois, Chicago, Sep. 2003, pp. 75-81.

[33] C. P. Lawrence, I. Kosuke, "Design Error Classification and Knowledge Management," *Journal of Knowledge Management Practice*, vol. 10, no. 9, May 2004, pp. 72-81.

[34] D. Leffingwell, "Calculating your Return on Investment from More Effective Requirements Management," *American Programmer*, vol. 10, no. 4, 1997, pp. 13-16.

[35] M. Lezak, E. D. Perry, D. Stoll, "A Case Study in Root Cause Defect Analysis," *International Conference on Software Engineering,* Limerick, Ireland, vol. 16, no.11, 2000, pp. 428-437.

[36] R. R. Lutz, "Analyzing Software Requirement Errors in Safety Critical, Embedded Systems," *IEEE International Symposium on Requirements Engineering,* IEEE Computer Society, Jan. 1993, pp. 126-133.

[37] R. R. Lutz, "Targeting Safety Related Errors During Software Requirement Analysis," *The Journal of Systems and Software*, vol. 18, no. 5, 1993, pp. 99-106.

[38] R. R. Lutz, K. Wong, S. Johnny, "Constraint Checking During Error Recovery," *NASA Technology 2002 Conference,* Dec. 1992, pp. 142-153; http://citeseer.ist.psu.edu/lutz96constraint.html.

[39] R. R. Lutz, M. R. Woodhouse, "Requirement Analysis Using Forward and Backward Search," *Annals of Software Engineering,* vol. 3, no. 1, May 1997, pp. 72-81.

[40] R. Matthias, "Why and What Can We Learn From Human Errors?" *Advances in Applied Ergonomics*, vol. 36, no. 3, 1993, pp. 827-830.

[41] D. Meister, "The Nature of Human Error," *IEEE Software*, vol. 12, no. 7, 1989, pp. 15-21.

[42] D. Norman, *Cognitive Engineering*, Lawrence Erlbaum Associates, Mahwah, New Jersey, 1986, pp. 31-61.

[43] A. A. Porter, L. G. Votta and V.R. Basili "Comparing Detection Methods for Software Requirements Inspections: A Replicated Experiment*"* *IEEE Transactions of Software Engineering*, vol. 21, no. 6, 1994, pp. 563-575.

[44] L. Prechelt, "Accelerating Learning from Experience: Avoiding Defects Faster," *IEEE Software*, vol. 18, no. 1, 2001, pp. 56-61.

[45] J. Rasmussen, A. M. Petersen and L. P. Goodstein, *Cognitive Systems Engineering,* J. Wiley, New York, 1994.

[46] P. R. Raymond, "Two Corpuses of Spreadsheet Errors," *Proceedings: 33rd Hawaii International Conference on System Sciences,* vol. 1, *2000,* pp. 1-8.

[47] J. Reason, *Human Error*, Cambridge University Press, Cambridge, England, 1990.

[48] *Requirement Specification for Tactical Separation Assisted Flight Environment*, Center for Experimental Software Engineering.

[49] *Research Methods and Statistical Resources*, http://www.georgetown.edu/departments/psychology/researchmethods/statistics/inferential.htm.

[50] S. Sakthivel, "A Survey of Requirements Verification Techniques," *Journal of Information Technology*, vol.6, 1991, pp. 68-79.

[51] P. Sawyer, G. Kotonya, *Software Requirements*, John Wiley and Sons, May 2001, p. 391.

[52] S. A. Scott, W. A. Douglas, "The Human Factors Analysis and Classification System-HFACS," *National Technical Information Service*, Springfield, Virginia 22161; http://www.nifc.gov/safety_study/accident_invest/humanfactors_class&anly.pdf.

[53] G. M. Schneider, J. Martin, W. Tsai, "An Experimental Study of Fault Detection in User Requirements Documents," *ACM Transactions on Software Engineering and Methodology*, vol.1, no.2, April 1992, pp. 188-203.

[54] F. Shull, "Developing Techniques for Using Software Documents: A Series of Empirical Studies," doctoral dissertation, University of Maryland, College Park, Maryland, Dec. 1998; www.cs.umd.edu/projects/SoftEng/ESEG/papers/postscript/shull_diss.ps.gz.

[55] F. Shull, I. Rus, and V. Basili, "How Perspective Based Reading can Improve Requirement Inspections," *Computer*, vol. 33, no. 7, July 2000, pp. 73-79.

[56] C. Smidts, "A Stochastic Model of Human Errors in Software Development: Impact of Repair Times," *Proceedings: 10th International Symposium on Software Reliability,* 1999, pp. 94-103.

[57] *Software Engineering Laboratory: Software Measurement Guidebook*, tech. report SEL-94-002, NASA/GSFC Software Engineering Laboratory, Greenbelt, Maryland, 1994.

[58] S. T. Steven, "TRACEr and TRACEr-lite: bridging the gap between incident investigation and performance prediction," *Journal of Safety and Reliability,* vol. 25, no. 2, 2005, pp. 62-81.

[59] S. T. Steven, K. Barry, "Development and Application of a Human Error Identification Tool for Air Traffic Control," *Applied Ergonomics*, vol. 33, no. 2, 2002, pp. 319-336.

[60] C. Smidts, "A Stochastic Model of Human Errors in Software Development: Impact of Repair Times," *Proceedings: 10th International Symposium on Software Reliability*, 1999, pp. 91-103.

[61] T. Thelin, P. Runeson, C. Wohlin, "An Experimental Comparison of Usage Based and Checklist Based Reading," *IEEE Transactions on Software Engineering*, vol. 29, no. 8, pp. 687- 704, Aug. 2003.

[62] C. Trevor, S. Jim, C. Judith, K. Brain, "Human Error in the Software Generation Process," 1994; http://www.branchlines.org.uk/Research/ Tread1.pdf

[63] R. Vaughn, Requirements Engineering Course CSE 8273, fall 2005, http://www.cse.msstate.edu/~cse8273/fall05/index.html.

APPENDIX A

FIRST TRAINING: TRAINING ON ERROR ABSTRACTION PROCESS

This appendix presents the training process on how to abstract errors (i.e., underlying mistakes) from the faults? This training was given to subjects during experiment run shown in Figure 8 and was called "*Training on Error Abstraction*".

A.1 Purpose and Resources

The purpose of the training was to teach students about abstracting errors or underlying mistakes from faults.

The resources required before conducting the training were Software Requirement Specification Document, Individual fault lists from each of the subject involved and procedure for abstracting causes of faults. The length of the training was around 30-40 minutes.

Training Procedure:

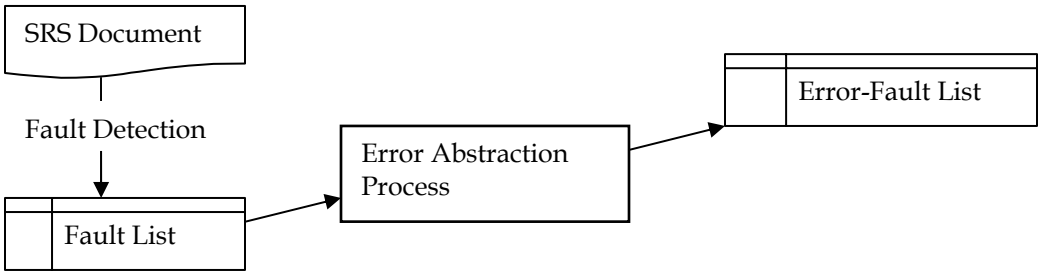


Figure A.1: Description of First Training Procedure

A.2 Guidelines for Abstracting Errors from Faults

The process of abstracting errors from the faults involved following steps:

Understand why each fault you identified represents a defect in a document?

First, think of nature of the fault that you have identified. Each fault represents part of requirement in which important information was left out, misstated, included in wrong Section or simply incorrect. Fault may also represent extraneous information that is provided but is not needed or used and may mislead the developers.

*Classification of the nature of faults***:**

After you identify each fault and its nature, you can classify the nature of faults under following categories:

Table A.2: Different Nature of Faults.

Correctness	Consistency	Traceable	Infeasible	Extraneous
Ambiguity	Completeness	Verifiable	Redundant	Wrong Section

*Grouping of isolated faults***:** After quantifying and classifying the basic nature, combine the isolated faults with similar pattern of information involved. Grouping of faults are based on the particular information common to them.

*Abstracting underlying mistake or error***:** Think of the underlying mistakes that might have caused these faults. If information is found in the wrong Section of the document, it might mean nothing more than that whoever wrote the requirement organized the requirement badly. On the other hand, if there is a contradiction within the document, then the requirements may reflect the underlying confusion about some of the functionality to be provided by the system.

Identification of the fundamental errors means finding patterns in collection of isolated faults. It is a creative process and not much of the guidelines can be provided. It

may be helpful to think about what kind of information is involved in the faults. Do particular functionalities, system users or performance requirements appear in multiple faults? This may indicate that these are particular aspects that are not well understood.

It may also be helpful to remember that not every fault is indicative of large error; some will undoubtedly result just from typos or misunderstandings of very small pieces of the functionality. Not every fault has to be a part of a pattern.

Example Application Systems

ATM: The purpose of the system is to support computerized banking network. The ATM network should not work independently and has to work with computers/software owned by bank. There are clearly defined interfaces for different systems.

AAD: AAD is an automatic ambulance dispatch system. This system describes the requirements for the computer aided dispatch system for the ambulance system.

PG: This system describes the requirement for parking garage control system. Its purpose is to supervise the entries in and out of the parking garage. The system allows or rejects entries into parking garage dependent on the available parking space.

Example 1:

1. ADD: The system will track incidents and raise warnings if the ambulance event crosses tolerances for incidents.
2. ADD: Regarding ambulance and incident states, the diagrams and error transitions are to be determined.
3. ADD: Reviewing the batch of incidents to determine the appropriateness of the service, however the appropriateness is undefined.

All requirements are incomplete since it is not described what the limits of tolerances are, some portions of SRS are to be determined and appropriateness is undefined for evaluating incidents.

Example 2:

1. ATM: The system operates independent of the network. The computers owned by the bank interfaces with ATM network and banks own cashier stations.

2. ATM: ATM should offer all kinds of transactions. The kind of transactions ATM offer is withdrawal.

3. ATM: The maximum limit for withdrawal is 600, and if withdrawal is greater then 1000 redo the transaction.

All of these requirements are inconsistent i.e. requirements of same system in different Sections contradict each other.

Example 3:

1. PGCS: the driver presses the button while any car leaves, then the driver gets the ticket. However, this causes "incorrect procedure" as we also need to check if number of parking spaces that are available (a) are greater than number of occupied non-reserved parking spaces (o).
2. PGCS: In process of checking a parking space if a>0. This is to provide every driver ticket at entrance only if there is parking space available. However, a>0 is wrong and needs to be a>0.
3. PGCS: Description of the formula is incorrect. It describes Maximum number of parking spaces in PG (k) = Number of parking spaces that are available (a)+

Number of reserved parking spaces in the PG(r); however k should be divided into o, a and r.

*Next Step***:** After you quantify the nature of the faults, the next step is to look for the patterns i.e. the information which could be functionality or any other requirement that appear in multiple faults and in different nature i.e. inconsistent or omission etc. Then, you have 2 context i.e. the "nature of that information" and the "information". After finding this pattern you start must think of the mistake that lead to these faults. Now, think of real mistakes that caused this fault.

*Logging of errors in "Error Report Form***"**: For each fault identified in the "Fault Report Form", write down the error you identified in the "Error Report Form".

Table A.3: Error-Report Form.

Error #	Page(s)	Fault Class	Fault #	Error Description	Time Found

APPENDIX B

SECOND TRAINING: TRAINING ON REQUIREMENT ERROR TAXONOMY

This appendix presents the training provided to the subjects involved in the experiment design (Figure 8) on how to use RET to classify errors and use them for subsequent fault detection in requirements document.

B.1 Purpose of the training and resources required

The purpose of this training is to investigate the usefulness of requirement error classification when applied to requirement document and its use on the fault detection density. The resources required for this training includes individual "Error-Fault" lists (output from 1st training), Description of "Requirement Error Classification Taxonomy", list of example errors for pilot run, procedure for application of "Requirement Error Classification" to find more faults. The length of the training was around 2 hours.

Training Procedure:

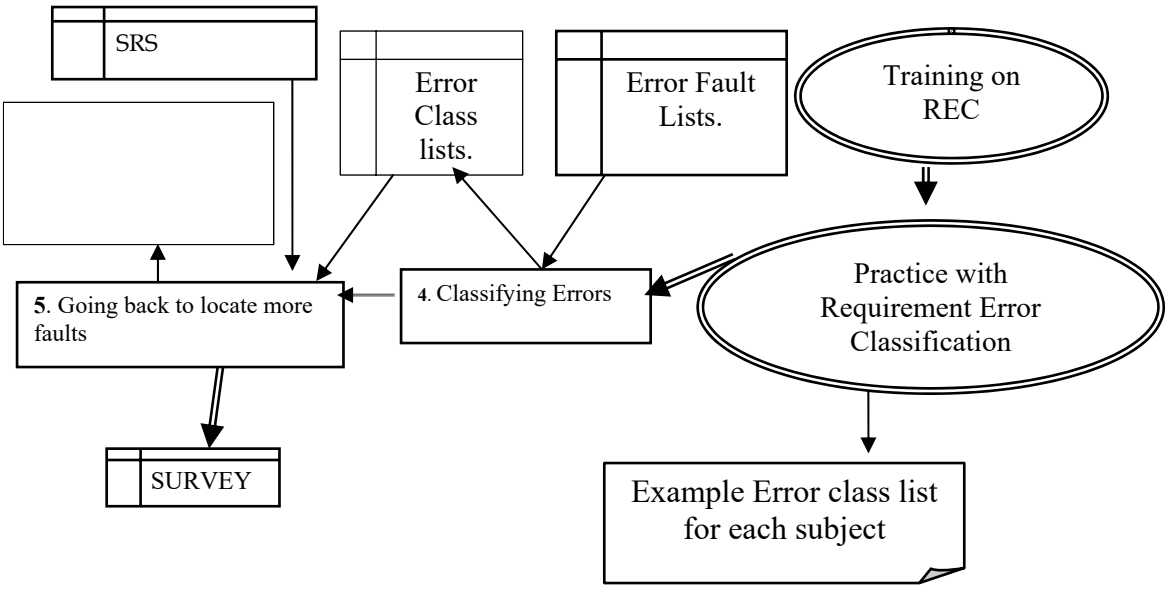


Figure B.1: Description of Second Training Procedure

B.2 Deliverables and Responsibilities:

Table B.1: Description of Tasks and Responsibilities

S. No.	Tasks / Responsibilities	Output
1.	Pilot run for classifying example errors into error classes in "Requirement Error classification".	Error- class lists for example errors.
2.	A) Classifying errors from "Error-Fault" lists into error classes described in "REC". B) Using "REC" to record more errors that might have occurred while developing SRS.	Error Class list for real errors.
3.	Usage of the error class list and looking back at each error in "Error Class List" to re inspect SRS for more faults.	Error-fault List

B.3 Description of training steps

B.3.1 Description of Requirement Error Classification Taxonomy

This step consists of describing our classification and describing all of the individual errors in each error class. Description of classification includes describing each error using the examples of errors and then describes the fault that can be caused by that error. The aim is to help subjects understand the nature and description of each error and to show them how different errors can lead to various kind of faults.

B.3.2 Classification of example errors into error classes from our classification

This step consists of measuring the understanding of the subjects regarding "Requirement Error Classification". The process is to have them practice with the classification of errors into error classes in our classification. The error classes used here are communication, participation, domain knowledge, understanding, application, inadequate selection of requirement plan, etc. We provided them with the example errors from application systems like ATM, PGCS and AAD. The subjects were told to use the information in the classification to classify the example errors into our error classes. We provided them with an "Error-Class" form and they will fill out that in the class. For eliminating any threat to the validity, we used two different lists i.e. A and B and distributed randomly. The errors in one list are reverse order in other list. The total number of errors were 12 and the subjects were given 30 minutes time for classifying the errors. We analyzed analyze whether the performance dips because of the number of errors or the difficulty in classifying them.

Table B.2: Error Class List for Classifying Example Errors.

Error #	Error Class	Reason	List A or B

After all the subjects fill this form, we would talk with them regarding the errors and their understanding. We would update their understanding and take the forms for analysis of effect of understanding of classification on their performance.

B.3.3 Developing "Error-Class list"

This step consists of two steps. The first step is to use the errors in "Error-fault" list from the first training to classify them into our 14 error classes i.e. communication, participation, domain knowledge, understanding specific application, process execution, other, inadequate selection of requirement plan, management, elicitation, analysis, traceability, organization, no usage of standard and specification errors. This process is same what subjects would do with the example errors in the class. They subjects used the following form. The field "Error #" is just the cross reference number, "Error Description" is same description as what you have in your Error-Fault list, "Error-Class" field is one of the 14 error classes that are in the classification taxonomy, and "Found prior to training 2" would be all yes for the errors you have already found.

Table B.3: Error Class List for Classifying Errors.

Error	Error Description	Error Class	Time taken	Found Prior to Training 2

After this is done, the next step was to go back and look at "Requirement Error Classification Taxonomy" to find more errors from the bullets (>) under each of the error class. The task was to think of these errors and see if the errors might have been committed. If so, then add them to error-class list. For example, while going through the bullets under communication you might discover that you have committed "Lack of communication of changes made to document". If so, add that error description and error

class to the list. Repeat this step for all error classes in the taxonomy. Now, the error class list would expand because more errors have been found using our taxonomy.

B.3.4 Developing new "Error-Fault" list

This step consists of looking at each of the error in the "Error- Class list" (from previous step) and uses that information to find more faults while re-inspecting the SRS document. Now, here the task was to find more and new faults that were not found before. Also one error can lead to multiple faults. The "Error-Fault" list used by subjects is as follows:

Table B.4: Error Fault List.

Error Number (From Error List)	Fault Caused	Fault Description	**Page** Number	Time taken

In this list, "Error number" field helps you to link it with error class list, "Fault Caused" field contained fault classes i.e. Correctness, Consistency, Traceable, Infeasible, Extraneous, Ambiguity, Completeness, Verifiable, Redundant. "Fault Description" field defines the fault in a manner that one should be able to understand and fix the fault.

APPENDIX C

QUESTIONNAIRE

Requirement Error Abstraction Process

Please underline/mark the scale

1. How confident are you in using it?

	1	2	3
Confidence	Low	Medium	High

2. How meaningful are the results produced by the abstraction process?

	1	2	3
Meaningful Results	Low	Medium	High

3. How much does it help you understand the real problems in requirements?

	1	2	3
Understanding	Low	Medium	High

Explain: _____

_____

_____

4. Was the effort spent in this process worthwhile? Also explain.

	1	2	3
Worthiness of effort	Not at all	Some	Very much

Answer: _____

_____

5. How Confident are you that the errors are a true representation of misunderstandings?

	1	2	3
Confidence in errors	Low	Medium	High

6. How long did you and your team spend in following "processes" and how closely you followed them?

1. **Abstraction of errors from Faults: 1st training:**

	1	2	3
Process Conformance	Low	Medium	High
Time spent (in minutes)			

2. Error Classification (Classifying errors into "Requirement Error Classes")

2nd training:

	1	2	3
Process Conformance	Low	Medium	High
Time spent(in minutes)			

Re - Inspecting SRS using "Requirement Error Classification" 2nd training:

	1	2	3
Process Conformance	Little	Medium	High
Time spent(in minutes)			

7. How did your team consolidate individual "Error" lists into a common team list?

Answer: _____

_____

_____

8. What problems did you encounter when using "Error Abstraction Process"? What can be done to overcome these problems?

Answer: _____

_____

_____

9. What is gained from the "Error Abstraction Process"?

Answer: _____

_____

_____

Requirements Error Classification taxonomy

1. Rate the following attributes regarding the "Requirement Error Classification Taxonomy" and explain?

Attributes	1	2	3
Simple	Low	Medium	High
Easy to understand	Low	Medium	High
Easy to use	Low	Medium	High
Intuitive	Low	Medium	High
Orthogonal	Low	Medium	High
Comprehensive	Low	Medium	High
Useful	Low	Medium	High
Uniformity across different products	Low	Medium	High

Explain:

2. Rate the following attributes:

Rank	1	2	3
Adequacy of error classes	Low	Medium	High
Ease of placing errors in an error class	Low	Medium	High

3. Do you think it will help to improve learning and communication among developers? Why or why not?

Answer: _____

_____

4. Rate the following attributes for future work on RET:

Attributes	1	2	3
Worthiness of developing error taxonomies for subsequent phases (e.g., design, testing etc).	Low	Medium	High
Usefulness of Deriving tools from RET for quality improvement	Low	Medium	High

5. What improvement or changes would you suggest for the "Requirement Error Classification Taxonomy"?

Answer: _____

_____

_____

Training

1. How useful was the training? Please rate them? Explain if it was not useful.

First Training

	1	2	3
Usefulness	Low	Medium	High

Second Training

	1	2	3
Usefulness	Low	Medium	High

Answer: _____

_____

_____

2. Was there anything missing from the training that would have helped you do a better job? If so, what?

Answer: _____

_____

_____

3. What will you do differently next time you use this taxonomy on requirements document or use this taxonomy for other phases?

Answer: _____

_____