APPLICATION OF HUMAN ERROR THEORIES IN DETECTING AND PREVENTING SOFTWARE REQUIREMENT ERRORS

by

WENHUA HU

JEFFREY C. CARVER, COMMITTEE CHAIR
JEFF GRAY
GARY BRADSHAW
TRAVIS ATKISON
RANDY SMITH

A DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in the Department of Computer Science
in the Graduate School of
The University of Alabama

TUSCALOOSA, ALABAMA

2017

ABSTRACT

Developing correct software requirements is important for overall software quality. Most existing quality improvement approaches focus on detection and removal of faults (i.e., problems recorded in a document) as opposed to identifying the underlying errors that produced those faults. Accordingly, developers are likely to make the same errors in the future and not recognize other existing faults with the same origins. The Requirement Error Taxonomy (RET) developed by Walia and Carver helps focus the developer's attention on common errors that can occur during requirements engineering. However, because development of software requirements is a human-centric process, requirements engineers will likely make human errors during the process which may lead to undetected faults. Thus, in order to bridge the gap, the goals of my dissertation are: (1) construct a complete Human Error Taxonomy (HET) for the software requirements stage; (2) investigate the usefulness of HET as a defect detection technique; (3) investigate the effectiveness of HET as a defect prevention technique; and (4) provide specific defect prevention measurements for each error in HET.

To address these goals, the dissertation contains three articles. The first article is a systematic literature review that uses insights from cognitive psychology research on human errors to develop formal HET to help software engineers improve software requirements specification (SRS) documents. After building the HET, it is necessary to empirically evaluate its effectiveness. Thus, the second article describes two studies to evaluate the usefulness of the HET in the process of defect detection. Finally, the third article analyzes the usefulness of HET for defect prevention and provides strategies for preventing specific errors in the SRS.

ACKNOWLEDGMENTS

I am pleased to have this opportunity to thank the many colleagues, friends, and faculty members who have helped me with this research project. I am most indebted to Jeffrey Carver, the chairman of this dissertation, for sharing his research expertise and wisdom regarding motivational theory. I would also like to thank all of my committee members, Jeff Gray, Travis Atkison, and Randy Smith for their invaluable input, inspiring questions, and support of both the dissertation and my academic progress. I would like to thank all of my group members who helped me at every step of my graduate research, with special thanks to Ahmed Al-Zubidy (graduate student), Amanda Lee (graduate student), Amiangshu Bosu (graduate student) for their help in providing feedback for my research. I also would like to thank to all my research project members including Gursimran Walia, Gary Bradshaw, and Vaibhav Anu for experiment design, data analysis, and paper writing.

Thanks to all the staff from the Computer Science department and CAPS, Kathy DeGraw, Kathleen Morris, Wendy Bryant, Valerie Trull. I am grateful for the financial support from National Science Foundation Awards 1423279 and 1421006.

Finally, I would like to thank my family and friends with all encouragements and support.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Software engineering, especially during the early phases, is a human-centric activity. Software engineers must gather customer needs, translate those needs to requirements, and validate the correctness, completeness, and feasibility of those requirements. Because of the involvement of various people in this process, there is the potential for mistakes to occur. The field of Human Error research in Psychology has dealt with the types of mistakes that people can make when doing various types of tasks. A Human Error occurs when a developer's thought process is flawed. These flaws can be the result of various causes. Human Error researchers have identified and classified the flaws so they can provide guidance on how to addresses them.

Software development is a human-centric process. To improve software quality, it is important to detect and prevent human errors during the development of software artifacts. Development of the software requirements specification (SRS) is the crucial first stage in the software engineering lifecycle because problems not identified or prevented propagate to subsequent phases where they are more difficult and expensive to fix. During the requirements stage, developers record the information gathered from different stakeholders into a natural language document because this process is human-based and because of the ambiguities of natural language, human errors are likely at this phase. Due to the large expense to find and fix problem after they occur, it would be much better to prevent problems in the first place. Having an understanding of these errors could reduce their potential of occurrence. Thus, in my

dissertation, I focused on how to help developers identify and prevent human errors in the requirements phase.

There are three terms that are important for my work. These terms have competing and contradictory definitions in the literature. For clarity, I use the following definition for each term, which are drawn from a software engineering book [1].

Error *– mistakes that happened in the process of making human thought. Each error may affect one or more elements, and may be caused by several different sources, such as incorrect execution of a planned action, or inaccurate and incomplete understanding of a system. In the context of SRS, the error means misunderstanding the actual needs of users or customers. According to the four definitions provided in IEEE standard [2], error includes both program error and human error; However, in my research, the definition of error is more closely to human error rather than program error.*

Fault *– specific demonstration of an error. Several faults may be caused by one error or some identified faults may be caused by various errors.*

Failure *– the operation behavior of a system is a departure from the user expectation, such as a software crash or some incorrect output. Usually, several faults may cause a particular failure and some faults may never cause any failure.*

Researchers have developed several fault-based, early-lifecycle quality improvement methods. The general idea of these methods is to group faults based on their characteristics into taxonomies. Studies showed that fault taxonomies can help developers significantly improve software quality [3, 4, 5, 6]. However, because fault-based techniques focus only on the faults, they do not help developers identify all kinds of faults and learn from these faults [7]. Thus, they do not help with defect prevention. Lawrence and Kosuke criticized these methods due to the

lack of some attributes to become an "ideal mistake-proofing taxonomy" that will provide guidance to developers. These attributes include collectively exhaustive, mutually exclusive, easily understood, simple, and less useful [8].

Error-based methods overcome the shortcomings of these faults-based techniques by helping developers identify the reasons why they inserted the faults. Especially the work of Lanubile [9], he developed an Error-Abstraction method to help inspectors abstract the faults to the errors that likely cause them. This knowledge should help prevent developers from making related faults in the future. However, the existing error-based techniques are not based on strong cognitive theories.

Based on the shortcomings of the existing fault prevention techniques, Walia & Carver extended Lanubile's work [10] by developing a taxonomy of requirements errors to support the process of error identification [11]. However, they derived this taxonomy from the published literature without involvement of a human error specialist. While the error classes were of sufficient utility to reduce errors, as evidenced by the results of their studies, this initial classification was never intended to be the complete and final product. Section 1.1 describes these approaches in more detail.

Motivated by Walia and Carver's requirements error taxonomy, the overall goal of my dissertation is to use insights from cognitive psychology research on human errors to develop a human error taxonomy (HET) and evaluate whether the HET is useful for detecting and preventing the faults and errors.

1.1. Background and related work

Defect prevention techniques can be used during the creation of software artifacts to help developers create high-quality artifacts. By preventing defects from occurring, these artifacts

should have fewer faults that must be removed during inspection and testing. Defect prevention techniques should focus developers' attention on common errors that can occur during requirements engineering. By focusing on those errors, the developers will be less likely to commit them. This section describes existing defect prevention techniques and their effect on software quality.

1.1.1 Error-based quality improvement methods

Many researchers have pointed out the advantages of error-based analysis techniques on improving software quality compared with the fault-based analysis approaches. Root cause analysis (RCA) [12], orthogonal defect classification (ODC) [13], and error abstraction [10] are the three most important error-based analysis approaches.

RCA [12] aims to help software engineers identify the underlying causes for development problems. It provides insight into the classification of software defects and introduces four multidimensional defect triggers for the different defect types. These triggers help developers identify the root cause of each defect to prevent these defects happening again. Because this method helps developers identify systematic problems during the testing phase, separate from the development process, it cannot provide valuable in-process feedback to developers to help prevent defects.

ODC [13] provides developers with in-process feedback on development activities. Developers use ODC to classify defects into a predefined taxonomy, identify the trigger that caused the defect to appear (not the cause of the defect injection), and then point out the software development process that needed to be resolved. Because this process is applied to code and the triggers explain the actions that revealed the failure, it is more objective than predicting the actual cause of defect insertion. But this technique only helps developers understand what caused a fault to be revealed (i.e., the failure) not what caused the fault to be inserted (i.e., the error).

Therefore, it is not able to help with defect prevention.

The error abstraction method [9] overcomes these shortcomings. The purpose of error abstraction is to help developers determine the underlying sources of faults detected in the inspection process, i.e., the error. Inspectors follow a process to abstract the faults to the errors that likely cause them, then use these errors to guide a re-inspection and detect other related faults that were overlooked during the original inspection. However, the error abstraction approach is subjective and highly dependent upon the abilities of the reviewer.

Based on the shortcomings of these approaches, in 2009, Walia and Carver conducted an SLR to extend the error abstraction technique by adding a requirements error taxonomy. During the SLR, they identified requirements errors described in the SE literature and human errors related to software requirements described in the human cognition and psychology literature and built a requirement error taxonomy (RET) [11]. The resulting error taxonomy had three high-level error types: People Errors (arise from fallibilities of the people involved in the development process), Process Errors (arise while selecting the appropriate processes for achieving the desired goals and relate mostly to the inadequacy of the requirements engineering process), and Documentation Errors (arise from mistakes in organizing and specifying the requirements, regardless of whether the developer properly understood the requirements), with 14 detailed error classes inside these error types. They also conducted a family of experiments to validate the taxonomy [14]. The results show that using structured error information can provide value to software requirements authors.

In the RCA, ODC, and Lanubile's error abstraction, developers analyze only a sample of faults, potentially overlooking many errors. These methods also lack a formal process to assist developers in finding and preventing errors. Walia and Carver's RET approach built a formal

error taxonomy, however, this taxonomy is just about common requirements errors. Meanwhile, a major drawback of these approaches is the lack of strong cognitive theory to describe the types of errors people make during development. Therefore, in this dissertation, I combine software engineering research with human error research to provide a more complete human error taxonomy to help developers detect and prevent the errors in the requirements phase.

1.1.2 Human errors and software engineering

Human errors have been examined in areas such as medicine, aviation, and the oil industry. Error specialists within the field of cognitive psychology have developed discipline-specific taxonomies of mental human errors that capture systematic failures in performance [15,16]. Across a variety of disciplines, such human mental error studies capitalized on basic theoretical research in human cognition, typically employing an information processing framework, which provides a coherent account of human errors when performing different tasks. It quickly became apparent that errors generally were not the result of irrational or maladaptive tendencies, but instead resulted from normal psychological processes gone awry.

Similarly, human mental errors in software engineering arise during information processing particularly as the system representation is translated from one form to another (i.e., from customer needs to requirements to architecture to design to code). Because software development is human-centric, many software failures can be traced back to human mental errors [17,18]. As a result, the application of human error research to software engineering offers great promise for reducing defects and improving the quality of software.

The software requirements stage is the first and most crucial stage in the software development lifecycle. The quality of software products largely depends on the quality of the underlying requirements. Previous research has helped underline the importance of developing correct requirements. First, Dethomad and Anthony found that defects injected in the

requirements stages are more difficult to identify and more expensive to fix than other defects [19]. Second, Chen and Huang (2009) concluded that documentation and requirements problems are among the top 10 high-severity problems [20]. Third, Hamill and Goseva-Popstojanova [21] showed that the majority of software failures are caused by defects in the requirements document. Therefore, it is crucial to identify the root cause of requirements defects and prevent the propagation of requirements defects to the later software stages.

In addition to the importance of developing correct requirements, other research has shown that human errors are the leading cause of software defects [20]. Problems arise from insufficient skills or experience [22] and from lack of proper training [23]. Given the importance of human error to the development of high-quality software requirements, it seems reasonable to focus research efforts here. There has been little prior research on applying human error concepts to software development process. Thus, my research focuses on identifying and classifying the errors that can occur during the development of the software requirements.

1.2. The Goal of This Research

Walia and Carver's RET was based on a software engineering literature review. Though this taxonomy can help developers identify requirements errors and improve software document, they did not directly interact with a human error expert during its development. Thus, this taxonomy was intended as an initial analysis, not as the last word on the topic. A more theoretically-based, refined, and detailed taxonomy is likely to better capture the cognitive human errors that can happen in the software requirements stages. Such a taxonomy will help identify and correct errors as well as prevent them from occurring in the first place. The overall goal of my dissertation is **to use insights from cognitive psychology research on human**

errors to develop a human error taxonomy (HET) and evaluate whether the HET is useful for detecting and preventing faults and errors.

To achieve this goal, the first objective of my dissertation is *to build a human error taxonomy (RO1, shown in Figure 1.1)*. By investigating the context of human errors and how humans make errors during the software requirements develop process, we will gain a deeper understanding of software development errors (and the resulting faults), informed by knowledge of the human cognitive mechanisms and how they fail. The second objective is to *validate the usefulness of the HET for defect detection and prevention (RO2, shown in Figure 1.1).*

1.3. Plan of action

This is an article-style dissertation consisting of three articles. Figure 1.1 shows the organization of the dissertation based on the three articles and how they address the research objectives.

Article 1 is a systematic literature review to identify and classify human errors in software requirements. The goals of this review are to:

- understand the types of human errors in published software engineering papers;
- understand the root causes of these errors;
- analyze the application of cognitive theory in the process of software development; and
- build a formal human error taxonomy to help developers identify and prevent errors and faults.

Working with a human error expert, we analyzed human error literature to understand how those theories can be applied to software engineering tasks. Then, we mapped the requirements error information from software engineering to the human error taxonomies from

Psychology as the basis for our own theoretically-grounded taxonomy. The results of this literature review address RO1.

Article 2 describes a classroom-based feasibility study and a practitioner study consisting of interviews and survey data to evaluate the usefulness of the HET for detecting human errors.

The goals of these studies are:

- **Validate HET**. I use empirical data from the classroom study to validate whether the HET can help students detect requirement errors and faults.
- **Evaluate the effect of the HET in industrial settings**. Using data from industrial practitioners, I analyzed whether the HET describes errors that occur in practice.

The results of this study help to address RO2.

Article 3 also describes a control group classroom study and a practitioner study consisting of interviews and survey data. The main goal of this article is to evaluate whether HET is useful to help developers prevent requirements faults and errors and to provide guidelines and advice about the prevention of each type of error. The results of the study help address RO2.

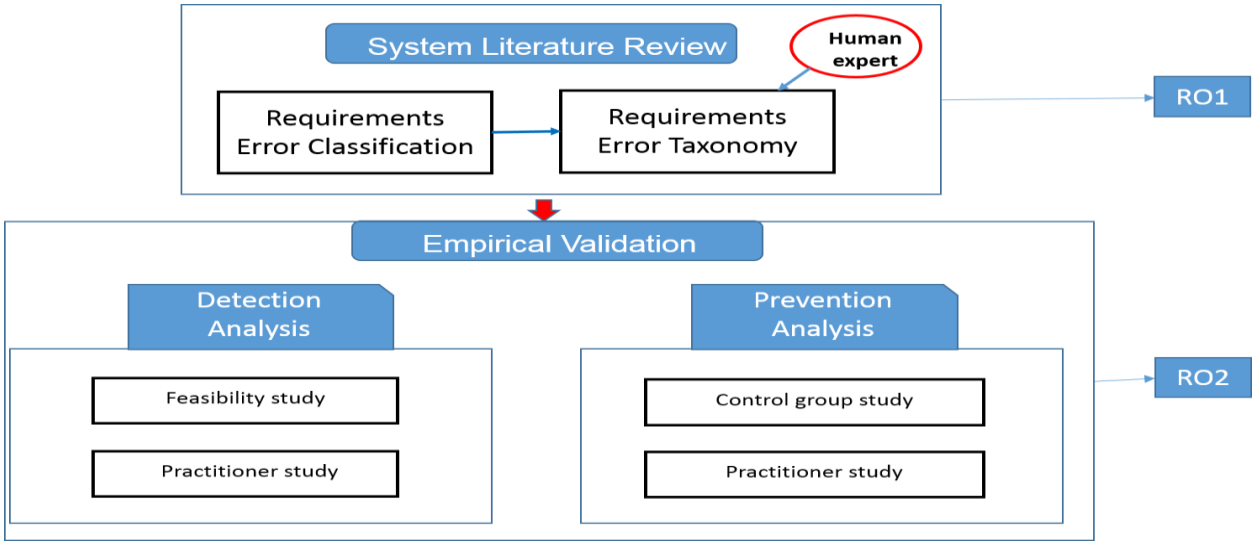


Figure 1.1. Research Process

1.4. Contributions

The main contribution of my dissertation are:

(1) Identification of reported human errors that occur during the requirements phase and organize the human errors into a human error taxonomy;

(2) Validation of the effectiveness of the human error taxonomy for detecting and preventing requirements errors and faults in classroom and industrial settings.

(3) Identification of prevention mechanisms for each HET error type to help developers inject fewer related errors and faults and improve the quality of requirements documents.

References

[1] Sommerville, ISoftware engineering, eighth ed., Addison Wesley, Harlow, England, 2007.

[2] IEEE std 610.12-1990. IEEE Standard Glossary of Software Engineering Terminology, 1990.

[3] Freimut, B., Denger, C., Ketterer. M.: An industrial case study of implementing and validating defect classification for process improvement and quality management. In Proceedings of 11th IEEE International Software Metrics Symposium, 2005: 9-19.

[4] Hayes. J. H.: Building a requirement fault taxonomy experiences from a NASA verification and validation research project. In proceeding of the 14th International Symposium on Software Reliability Engineering, 2003: 49-60.

[5] Masuck, C: Incorporating a fault categorization and analysis process in the software build cycle. Journal of Computer Sciences in Colleges, 2005, 20(5): 239-248.

[6] Carver, J. C.: Impact of background and experience on software inspection. Ph. D. Thesis, Department of Computer Science, University of Maryland, 2003.

[7] Westland, J. C.: The cost of errors in software development: evidence from industry. The Journal of Systems and Software, 2002, 62: 1-9.

[8] Lawrence, C. P., Kosuke, I.: Design error classification and knowledge management. Journal of Knowledge Management Practice, 2004, 10(9): 72-81.

[9] Lanubile, F., Shull, F., Basili, V. R.: Experimenting with error abstraction in requirements documents. In Proceedings of Fifth International Software Metrics Symposium, METRICS'98, IEEE Computer Society, Bethesda, MD, 1998: 114-121.

[10] F. Lanubile, F. Shull, V. R. Basili. Experimenting with error abstraction in requirements documents. In Proceedings of Fifth International Software Metrics Symposium, METRICS'98, IEEE Computer Society, Bethesda, MD, 1998, PP. 114-121.

[11] Walia, G. S., Carver, J. C.: A systematic literature review to identify and classify software requirement errors. Information and Software Technology, 2009, 51: 1087-1109.

[12] Leszak, M., Perry, D. E., Stoll, D. A case study in root cause defect analysis. In Proceedings of the 22nd International Conference on Software Engineering, ACM press, Limerick, Ireland, 2000: 428-437.

[13] Chillarege, R., Bhandari, I. S., Chaae, J. K., Halliday, M. J., Moebus, D. S., Ray, B. K., Wong, M. Y.: Orthogonal defect classification - a concept for in-process measurement. IEEE Transactions on Software Engineering, 1992, 18(11): 943-956.

[14] Walia, G. S., Carver, J. C.: Using error abstraction and classification to improve the quality of requirements: conclusions from family of studies. Empirical Software Engineering, 2013, 18(4): 625-658.

[15] Berra,Y.: Error in medicine: what have we learned? Annals of Internal Medicine, 2000, 132(9): 763-767.

[16] Wiegman, D. A., Shappell, S. A.: Human error perspectives in aviation. The International Journal of Aviation Psychology, 2000, 11(4): 341-357.

[17] Trevor, C., Jim, S., Judith, C., Brain. K.: Human error in software generation process, University of Technology, Loughborough, England, 1994.

[18] Lopes, M., Forster, C.: Application of human error theories for the process improvement of requirements engineering. Information Sciences, 2013, 250: 142-161.

[19] Anthony, D.: Technology requirements of integrated, critical digital flight systems. In: AIAA Guidance, Navigation and Control Conference, Monterey, CA, Technical Papers, American Institute of Aeronautics and Astronautics, New York, 1987, 2: 1579-1583.

[20] Chen, J. C., Huang, S. J.: An empirical analysis of the impact of software development problem factors on software maintainability, Journal of Systems and Software, 2009, 82: 981-992.

[21] Hamill, M., Katerina, G. P.: Common trends in software fault and failure data, IEEE Transaction on Software Engineering, 2009, 35: 484-496.

[22] Jiang, J. J., Klein, G.: Software development risks to project effectiveness. The Journal of Systems and Software, 2000, 52 (1): 3-10.

[23] Han, W. M., Huang, S. J. An empirical analysis of risk components and performance on software projects. The Journal of Systems and Software 2007, 80 (1), 42-50.

CHAPTER 2

DEVELOPMENT OF HUMAN ERROR TAXONOMY FOR SOFTWARE REQUIREMENTS: A SYSTEMATIC LITERATURE REVIEW

2.1. Introduction

Software engineering, especially during the early phases, is a human-centric activity. Software engineers must gather customer needs, translate those needs into requirements, and validate the correctness, completeness, and feasibility of those requirements. Because of the involvement of various people in this process, there is the potential for human errors to occur. Cognitive Psychology researchers have studied the types of errors people make when performing different types of tasks. In this paper, we apply the findings from this Human Error research to analyze and classify the types of human errors developers can make during the requirements engineering process.

Because software engineering researchers often have competing definitions for the same terms, it is important to clarify our terminology at the beginning. Based on IEEE Standard 24765 (ISO/IEC/IEEE 24765:2010), we use the following definitions through this paper:

- *Error (also referred to as Human Error)* – A mental error, i.e., the failing of human cognition in the process of problem solving, planning, or execution.
- *Fault* – The manifestation of an error recorded in a software artifact.
- *Failure* – The incorrect execution of a software system, e.g., resulting in a crash, unexpected operation, or incorrect output.

The chain from human error to fault to failure is not unbroken and inevitable. Generally, developers detect and correct the faults without investigating the underlying errors or may use software testing to reveal system failures that are repaired. Nevertheless, many system failures originate in a human error, which needs to be investigated for detecting and fixing software faults and failures. By one estimate, software failures (which often find their origination in mental errors) cost the industry $60 billion/year (Tassey 2002).

The Aviation and Nuclear Power domains also faced similar situations: Errors might originate in the mind of a member of the cockpit crew or in the mind of a reactor operator, but produce a fault that results in a plane crash or a reactor core meltdown. In both fields, human errors created problems measured not only in billions of dollars per year, but also in lives lost. By enlisting the aid of human error specialists, both fields have been able to achieve dramatic improvements in safety and efficiency. The process entailed (1) analyzing incident reports to reveal the human error(s) made; (2) identifying common error patterns; and (3) organizing those errors into a hierarchical error taxonomy. By examining the most frequent and costly errors in the hierarchy, investigators developed and implemented mediation strategies, thereby reducing the frequency and severity of errors (Diller et al., 2014; Scherer et al., 2010; Wiegmann and Detwiler 2005). Our premise in this paper is that a similar approach could be beneficial in the software engineering domain.

The software requirements phase is largely human-centric. Requirements analysts must interact with customers or domain experts to identify, understand, and document potential system requirements. Numerous studies have shown that it is significantly cheaper to find and fix requirements problems during the requirements activities than it is to find and fix those problems during downstream software development activities. Therefore, it is important to focus attention

on approaches that can help developers both improve the quality of the requirements and find requirements problems early.

Human error research can be particularly beneficial in this scenario. The high level of interaction and understanding required at this step of the software lifecycle can result in a number of problems. The application of concepts from human error research will allow us to categorize the types of errors that occur. Such a categorization will have two benefits. First, it will provide a framework to help requirements engineers understand the types of mistakes that can occur as they are developing requirements. Awareness of these potential problems should lead them to be more careful. Second, the categorization of requirements errors will help reviewers to ensure that requirements quality is sufficient to allow development to proceed.

Recognizing the potential benefit that a formalized error taxonomy could provide to the requirements engineering process, a subset of the current authors conducted a Systematic Literature Review to identify errors published in the literature from 1990-2006. Using a grounded theory approach (Glaser and Strauss 1967), they classified those requirement errors and organized them into a Requirement Error Taxonomy to support the detection and prevention of errors and resulting faults (Walia and Carver 2009). They performed that review without reference to contemporary psychological theories of how human errors occur. In this current review, we extend the previous review in two key ways. First, rather than developing a requirement error taxonomy strictly 'bottom-up' with no guiding theory, we engage directly with a human error researcher and use human error theory to ensure that we only include true human errors and organize those errors based on a standard human error classification system from cognitive psychology. Second, this review includes additional papers that have been published since the first review was completed.

To guide this literature review, we focused on the following research questions:

RQ1: What types of human errors that occur during the requirements engineering process can we identify from the software engineering and psychology literature?

RQ2: How can we organize the human errors identified in RQ1 into a taxonomy?

The primary contributions of this work include:

- Adapting human error research to a new domain (software quality improvement) through interaction between software engineering researchers and a human error researcher;
- Development of a systematic human error taxonomy for requirements to help requirements engineers understand and avoid common human errors made during the requirements engineering process.
- An analysis of the literature describing human errors that occur when working with software requirements to provide insight on whether a community is forming around common ideas.-

2.2. Background

Formal efforts to improve software quality span decades (Chillarege et al., 1992; Leszak et al., 2000). This section briefly reviews some of these efforts as preface to a more comprehensive discussion of the role of human error research and the contributions it can make in identifying, understanding, correcting, and avoiding errors in software engineering. Given the importance of catching and correcting errors and faults early in the software development process, we emphasize the requirements phase of the process in our efforts.

2.2.1 Historical perspectives on software quality improvement

Early research to improve software quality was focused on faults, or the manifestation of an error. Faults are sometimes identified by software failures (the termination of the ability of a product to perform a required function or its inability to perform within previously specified

limits). Because a fault made early in software development may lead to a later series of faults, researchers developed Root Cause Analysis (RCA) (Mays et al., 1990; Leszak et al., 2000) to identify the first fault in a (potentially lengthy) chain. RCA asks developers to trace faults back to their origin. Through this process, developers identify the triggering fault and can make modifications to procedures to reduce or eliminate the original fault. The goal is to prevent the first fault, thereby eliminating the entire fault chain.

Because RCA is a time-consuming enterprise, researchers developed the Orthogonal Defect Classification (ODC) to provide structure to the fault tracing process (Chillarege et al., 1992). Developers use ODC to classify defects and then identify the trigger that causes the defect to surface (not necessarily the cause of defect insertion). This approach accommodates a large number of defect reports and identifies the actions that revealed the failure. ODC avoids the tedious work of tracing faults back to their origins by focusing on the action that caused defects to be manifested, reducing the time and effort compared to RCA.

Both RCA and ODC are retrospective techniques that begin with fault reports: faults must be made and detected in order to trigger the investigative processes. ODC relies on statistical analyses to help identify the source fault, and so depends on a large and robust set of fault reports. Because they are retrospective, both RCA and ODC occur late in the development process, after errors and faults have accumulated and are more expensive to fix. In addition, neither approach helps to identify the mental errors made by software developers that caused the source fault in the first place. Thus, these approaches focus more on treating the symptoms (manifestations of error) rather than the true underlying cause (i.e., error).

Noting that it can be difficult to define, classify, and quantify requirements faults, Lanubile, Shull, and Basili proposed shifting the emphasis from faults to errors. They defined the

term error as a defect or flaw in the human thought process that occurs while trying to understand given information, while solving problems, or while using methods and tools (Lanubile et al., 1998). Such errors are the cause of faults. This shift is important because it helps understand why the error happened that led to the fault. Here they address the underlying cause (i.e., the error), not merely the symptoms (i.e. the faults). This approach represents a considerable advance over RCA and ODC.

Errors and faults have a complex relationship: One error may cause several faults, or one fault may spring from different errors. Consequently, it is no trivial matter to identify the error behind the fault. Lanubile, Shull, and Basili asked software inspectors to analyze the faults detected during an inspection to determine the underlying cause, i.e., the error. The inspectors then used the resulting list of errors to guide a re-inspection of software artifacts to detect additional faults that might have been overlooked during the original inspection. This re-inspection produced only a moderate improvement in fault detection, an average of 1.7 per participant (Lanubile et al., 1998).

This error abstraction methodology is, once again, a retrospective process dependent upon first making an error that produced a fault, then detecting the fault to initiate a trace-back process to recover the error. Lanubile, Shull, and Basili also observed, from their experience as reviewers, that requirements errors are generally domain-specific, i.e., errors often relate to specific functionalities in the system rather than to general causes that may apply to any system (Lanubile et al., 1998). The implication is that cataloging and organizing errors, being idiosyncratic to each software development process, will not help other software development efforts. Such idiosyncrasy would limit the generality of error abstraction: the error abstraction would need to be performed independently for each software development effort. In addition,

this process is retrospective and can only be employed after errors have produced faults that have been detected. Retrospective fault detection techniques leave room for faults to go undetected and propagate into later phases of software development, increasing the time and cost of fixing the system. Therefore, there is still room for techniques that can both prevent faults as well as help in early identification of faults.

2.2.2 A cognitive psychology perspective on software quality improvement

Most psychologists reject the view that errors are idiosyncratic and strongly dependent on context. James Reason, a noted authority on human error, rejected the popular notion that "errors occur 'out of the blue' and are highly variable in their form" as a myth. Reason proposed that errors are not random and that their occurrence takes recurrent and predictable forms (Reason 2008). This predictability of errors is a crucial issue: To the extent that software engineers make the same errors across different projects, we can identify errors from completed projects and employ this understanding in new ones. In such cases, developers can use errors proactively to search for faults before they have been identified through other means and preventatively to modify methods of software development in such a way as to reduce or eliminate errors from happening in the first place.

To identify the types of errors that occur commonly across projects, Walia and Carver conducted a systematic literature review of software requirements errors. They identified 119 requirements errors from software engineering and psychology literature. Employing a bottom-up approach, Walia and Carver developed a Requirement Error Taxonomy (RET) by grouping common errors together into fourteen error classes. They then grouped these error classes into three high-level error types: People Errors (arising from fallibilities of the people involved in the development process), Process Errors (arising while selecting the appropriate processes for achieving the desired goals and relating mostly to the inadequacy of the requirements

engineering process), and Documentation Errors (arising from mistakes in organizing and specifying the requirements, regardless of whether the developer properly understood the requirements) (Walia and Carver 2009).

Walia and Carver used four experiments to investigate the utility of the RET (Walia and Carver 2013). Their work incorporated an inspection-abstraction-reinspection paradigm, similar to the one used by Lanubile, Shull, and Basili. Participants first analyzed their requirements documents to locate faults, then they analyzed those faults to abstract errors, followed by a reinspection of the document guided by the identified errors. The inspectors used the RET to guide the error abstraction process, as opposed to the ad hoc approach in Lanubile, Shull, and Basili's work. The reinspection step resulted in a significant number of new faults identified. Subjectively, participants indicated that the structured error information in the RET helped them understand their errors.

The taxonomic structure employed by Walia and Carver drew upon the published software engineering literature, but is strikingly different from psychological organizations of human errors. The RET includes a category of errors made by people, another category that ascribes errors to different processes, while the third localizes errors in documentation. Clearly all of the errors that were observed were committed by people, most happened while following or attempting to follow the software engineering process, and resulted in a fault somewhere in the project documentation. Hence, a step further from RET would be to divide the RET errors into separate and distinct human error categories. This observation motivated us to consider that a stronger connection to the rich psychological literature on human errors might reveal a more useful error taxonomy which could be applied to requirements engineering.

2.2.3 Reason's classification of human errors

James Reason developed one of the most comprehensive psychological accounts of human errors (Reason 1990). Central to his account is the construct of a plan. Loosely speaking, a plan is a series of actions that are needed to reach a goal. Three plan-based errors are slips, lapses, and mistakes. Assume that our goal is to drive to the store and entails steps such as starting the car, backing down the driveway to the street, navigating the route, and parking in the store lot. If we put the wrong key in the ignition, we have committed a failure of execution known as a slip. If we forget to put the transmission into reverse before stepping on the gas, the omitted step is a lapse. Failing to take into account the effect of a bridge closing and getting caught in traffic is a planning mistake.

Slips and lapses are execution errors, where an agent fails to properly carry out a correctly planned step. These are typically caused by inattention and memory failures, respectively. A mistake arises when the plan itself is inadequate. Reason associated errors with specific cognitive failings, such as inattention, memory failures, and lack of knowledge. If the cognitive failing can be identified ("I wasn't paying attention when I wrote this down"), we have a strong clue to the specific error that resulted. Accordingly, error identification can be undertaken even by those with little formal training in human psychology.

Reason's taxonomy has found diverse application in medical errors (Zhang et al., 2004), problems with Apollo 11 (Reason 1990), driver errors (Reason et al., 1990), and errors that led to the Chernobyl nuclear power plant meltdown (Reason 2008). The next section describes how we developed a new error taxonomy that was congruent with Reason's Slips, Lapses, Mistakes framework.

2.3. Research method

Table 2.1. Research Questions

#	Research Question
RQ1	What types of human errors that occur during the requirements engineering process can we identify from the software engineering and psychology literature?
RQ2	How can we organize the human errors identified in RQ1 into a taxonomy?

This review is a replication and extension of Walia and Carver's SLR (Walia and Carver 2009) that developed the RET (hereafter, referred to as SLR-2009). The current SLR expands upon the results of SLR-2009 by leveraging the cognitive theory of human errors to develop a formalized taxonomy of human errors that occur during the requirements phase of the software lifecycle. In this review, we employ a rigorous human error identification methodology that leads to a more formalized taxonomy that appropriately captures the sources of requirements faults, i.e., human mental errors. In this review, we followed the traditional systematic literature review (SLR) approach (Kitchenham 2004). To increase the rigor of the current review, the first and second authors (subsequently referred to as "researchers") independently performed each SLR step and met to resolve any disagreements.

2.3.1 Research questions

To guide the development of the human error taxonomy for requirements errors, we defined the high-level research questions in Table 2.1.

2.3.2 Source selection and search

To identify relevant papers, we used the same search strings as used in SLR-2009 (see Appendix A). We executed the search strings in: IEEExplore, INSPEC, ACM Digital Library, SCIRUS (Elsevier), Google Scholar, PsychINFO (EBSCO), and Science Citation Index, to ensure coverage of topics in software engineering, empirical studies, human cognition, and psychology. Because SLR-2009 included papers published through 2006, we only searched for

papers published after 2006 (through October 2014). Appendix A provides the detailed search strings, as adapted for each database. After the database search, we used snowballing to ensure a complete and comprehensive primary study selection process.

2.3.3 Study selection: inclusion and exclusion criteria

During the title-elimination phase, the two researchers (me and Anu Vaibhav) identified 144 and 136 papers, respectively (see Figure 2.1). Then, each researcher read the abstracts and keywords to exclude any papers that were not clearly related to the research questions. When in doubt, they included a paper until the next stage. At the end of this stage, the researchers' lists contained 97 and 88 papers, respectively.

Next, the researchers consolidated their lists into one master list. They added the 32 papers in common between their lists to the master list. For the remaining 216 papers (112 and 104, respectively), each researcher examined the papers on the other researcher's list to determine whether they agreed on inclusion. In cases where the other researcher agreed, they added the paper to the master list. In the remainder of the cases the researchers discussed and reached an agreement on whether to include the paper in the master list. This step resulted in 96 agreed-upon papers on the master list.

Next, the researchers divided the 96 papers between them for full-text review. Each reviewer used the inclusion/exclusion criteria in Table 2.2 to examine the full-text of 48 papers. They each made two lists, an include list and an exclude list. Any paper on the include list stayed in the study. Any paper on the exclude list was reviewed by the other researcher to make a final decision. The researchers discussed any disagreements and resolved them. This step resulted in 34 papers remaining on the included list.

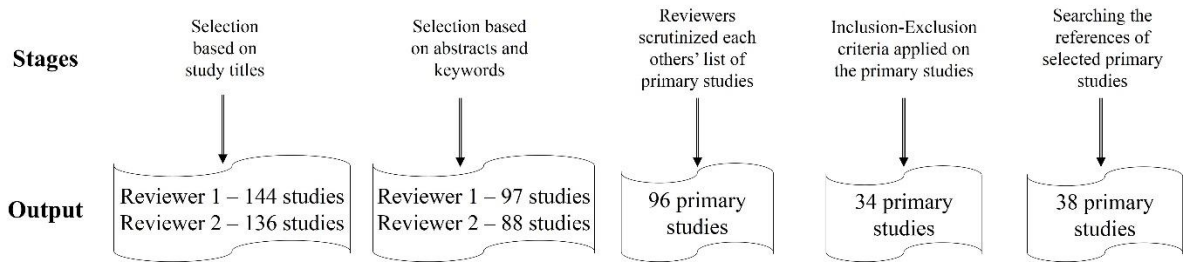


Figure 2.1. Primary study search and selection process

Finally, the researchers identified four additional papers through snowballing. The researchers did not find these papers during the initial database search because they included "risk components" and "warning signs" in the title rather than the more standard terms included in the

Table 2.2. Inclusion and exclusion criteria

RQ	Inclusion Criteria (Specific to RQ's)	Exclusion Criteria (same for both RQ's)
1	• Papers that focus on using human errors for improving software quality • Empirical studies (qualitative or quantitative) of using human error information in software development lifecycle • Papers that provide errors, mistakes or problems in the software development lifecycle. • Papers that provide error, fault, or defect classifications • Empirical studies (qualitative or quantitative) that provide causal analysis or root causes of software defects	• Papers based only on expert opinion • Short-papers, introductions to special issues, tutorials, and mini-tracks • Papers not related to any of the research questions • Preliminary conference versions of included journal papers • Studies whose findings are unclear and ambiguous
2	• Papers from psychology literature about models of human error • Papers from cognitive psychology literature about human thought process, planning, human reasoning or problem solving • Empirical studies (qualitative or quantitative) on human errors • Papers describing various human error classification systems	

search strings (Appendix A). This step resulted in 38 papers on the master list. Figure 2.1 summarizes the search and selection process.

The 38 papers were distributed across journals (15), conferences (19), workshops (3), and book chapters (1). In addition, these papers cover a wide spectrum of research topics, including: Computer Science, Software Engineering, Medical Quality, Healthcare Information Technology, Cognitive Psychology, Human Computer Interaction, Safety/Reliability Sciences, and Cognitive Informatics. Interestingly, the 38 papers appeared in 36 venues. That is, except for two cases (*Journal of Systems Software* and *Information Systems Journal*), each paper came from a unique publication venue. This result suggests that the research about human errors in software engineering is widely scattered and there is not yet a standard venue for publishing such research results. Section 2.5.3 discusses this observation in more detail. Appendix B contains the full distribution of papers across publication venues.

Table 2.3. Common Data Items for Extracting Information

Data item	Description
Study Identifier	Unique identifier for the paper (same as the reference number)
Bibliographic data	Author, year, title, source
Type of article	Journal/conference/technical report
Focus of Area	The field in which the research was conducted e.g., Software Engineering or Industrial Engineering or Psychology or Aviation or Medicine
Study aims	The aims or goals of the primary study
Context	Context relates to one/more search focus, i.e., research area(s) the paper focusses upon.
Study type	Industrial experiment, controlled experiment, survey, lessons learned.
Unit of analysis	Individual developers or department or organizational
Control Group	Yes, no; if ‘‘Yes”: number of groups and size per group
Data collection	How the data was collected, e.g., interviews, questionnaires, measurement forms, observations, and discussion
Data analysis	How the data was analyzed; qualitative, quantitative or mixed
Concepts	The key concepts or major ideas in the primary studies
Higher-order interpretations	The second- (and higher-) order interpretations arising from the key concepts of the primary studies. This can include limitations, guidelines or any additional information arising from application of major ideas/concepts
Study findings	Major findings and conclusions from the primary study

2.3.4 Data extraction and synthesis

Table 2.4. Data Items Relative to Each Search Focus

Search Focus	Data Item	Description
Quality Improvement Approach	Focus or process	Focus of the quality improvement approach and the process/method used to improve quality
	Benefits	Any benefits from applying the approach identified
	Limitations	Any limitations or problems identified in the approach
	Evidence	The empirical evidence indicating the benefits of using error information to improve software quality and any specific errors found
	Error focus	Yes or No; and if ‘‘Yes”, classify the solution foundation (next data item)
	Mechanism or Solution Foundation	1. Ad hoc - just something the investigators thought up but could have been supported by empirical work showing its effectiveness; or 2. Evidence-based - a notation of a systematic problem in the software engineering process that leads to a specific remediation method; or 3.Theory-based - draws support from research on human errors
Requirement Errors	Problems	Problems reported in requirement stage
	Errors	Reported errors (if provided in the paper) at the requirements stage
	Faults	Faults (if any information provided) at requirement stage
	Mechanism	Process used to analyze or abstract requirements errors (select one of the following): 1. Ad hoc - just something the investigators thought up but could have been supported by empirical work showing its effectiveness; or 2. Evidence-based — a notation of a systematic problem in the software engineering process that leads to a specific remediation method; or 3. Theory-based — draws support from research on human errors
Error-fault-defect taxonomies	Focus	The focus of the taxonomy (i.e., error, fault, or failure)
	Error Focus	Yes or No; if ‘‘Yes”, What was the process used to classify errors into a taxonomy?
	Requirements phase	Yes or No (whether it was applied in requirements phase)
	Benefits and Limitations	Benefits and/or Limitations of the taxonomy

	Evidence	The empirical evidence regarding the benefits of error/fault/defect taxonomy for software quality
Software inspections	Focus	The focus of inspection method (i.e., error, fault of failure)
	Error Focus	Yes or No; if ‘‘Yes”, how did it focus reviewers’ attention to detect errors during the inspection
	Requirement phase	Yes or No (Did it inspect requirements documents?)
	Evidence	The empirical evidence regarding the benefits/limitations of error-based inspection method
Human errors	Human errors and classifications	Description of errors made by human beings and classes of their fallibilities during planning, decision making and problem solving
	Evidence	The empirical evidence regarding errors made by humans in different situations (e.g., aircraft control) that are related to requirements errors

Based on publication venue and the type of information contained in the paper, we classified 11 papers as Cognitive Psychology papers and 27 as Software Engineering papers. The Cognitive Psychology papers focused primarily on models of human errors and human error taxonomies. The Software Engineering papers focused primarily on errors committed during the software development process. Because the focus of these types of papers differed, we extracted different types of data, as explained below.

To improve compatibility with SLR-2009, we modeled our data extraction form based on the one used in SLR-2009. We added a small number of data items to ensure that our focus was on human cognitive failures or human errors. We reanalyzed the SLR-2009 papers to extract this new information to ensure we had consistent information from all papers. Table 2.3 lists the data extracted from all primary studies. Table 4 lists the data extracted based on each primary study’s research focus.

Table 2.5. Human errors identified in the literature

Error #	Error Name	Source
1	Problem representation error	Huang et al., 2012
2	RE people do not understand the problem	Lehtinen et al., 2014
3	Assumptions in grey area	Kumaresh and Baskeran 2010
4	Wrong assumptions about stakeholder opinions	Lopes and Forster 2013
5	Lack of cohesion	Lopes and Forster 2013
6	Loss of information from stakeholders	Lopes and Forster 2013
7	Assumption that insufficient requirements are ok	Lehtinen et al., 2014
8	Low understanding of each other's roles	Bjamason et al., 2011
9	Not having a clear demarcation between client and users	Kushwaha and Misra 2006
10	Mistaken belief that it is impossible to specify non-functional requirements in a verifiable form	Firesmith 2007
11	Accidentally overlooking requirements	Firesmith 2007
12	Ignoring some requirements engineering tasks	Firesmith 2007
13	Inadequate Requirements Process	Firesmith 2007
14	Mistaken assumptions about the problem space	SLR-2009
15	Environment errors	SLR-2009
16	Information Management errors	SLR-2009
17	Lack of awareness of sources of requirements	SLR-2009
18	Application errors	SLR-2009
19	Developer (requirements developer) did not understand some aspect of the product or process	SLR-2009
20	User needs not well-understood or interpreted by different stakeholders	SLR-2009
21	Lack of understanding of the system	SLR-2009
22	Lack of system knowledge	SLR-2009
23	Not understanding some parts of the problem domain	SLR-2009
24	Misunderstandings caused by working simultaneously with several different software systems and domains	SLR-2009
25	Misunderstanding of some aspect of the overall functionality of the system	SLR-2009
26	Problem-Solution errors	SLR-2009
27	Misunderstanding of problem solution processes	SLR-2009
28	Semantic errors	SLR-2009
29	Syntax errors	SLR-2009
30	Clerical errors	SLR-2009
31	Carelessness while documenting requirements	SLR-2009

To ensure that both researchers had a consistent interpretation of the data items, we conducted a pilot study by having each researcher independently extract data from ten randomly selected papers. A comparison of the data extraction forms revealed a few minor inconsistencies in the interpretation of some data fields. The researchers discussed these issues to ensure they had a common understanding of the data items. They then proceeded to independently extract data from the remaining 28 papers. After extracting data, the researchers discussed and resolved any discrepancies to arrive at a final data extraction form for each paper.

2.4. Reporting the review

This section is organized around the answers to the two research questions listed in Section 2.3.

2.4.1 Research question 1 - What types of human errors during the requirements engineering process can we identify from the software engineering and psychology literature?

We began by extracting individual errors, error categories, sub-categories, error descriptions, and root causes from the 38 papers. Similarly, we reviewed the published SLR-2009 paper and extracted the errors and their descriptions (referring back to the original papers where necessary). An examination of this information revealed different interpretations of the term *error*. These interpretations included: program errors, software defects, requirements defects, end-user (or user) errors, requirements problems, and human error.

Before analyzing the *errors* to build a taxonomy, we had to determine whether each of these *errors* was truly a *human error*. Independently, the two researchers examined each error name and description to identify *human errors*. As before, they compared their results and resolved any discrepancies. The two reviewers had disagreements less than 5% of the time. Human error expert Bradshaw helped resolve these disagreements. This process resulted in 33 *human errors*. We eliminated two of those because they did not focus on requirements, bring the

total to 31 *human errors* in the requirements phase. Table 2.5 lists the 31 human errors identified in the literature sorted by the errors identified from the new set of papers followed by the errors identified from SLR-2009.

At this point, we also updated the list of papers included in the SLR. First, only seven of the twenty-seven software engineering papers identified in the search contained a true human error. Therefore, we removed the other 20 SE papers from further consideration. Conversely, none of the cognitive psychology papers contained a *true requirements engineering human error,* as confirmed by human error expert Bradshaw. Some of these papers discussed decision making errors or operator errors (which are different from software engineering errors). We excluded these papers for answering RQ1. In addition, there were eleven studies from SLR-2009 that identified human errors. Adding these paper sets together, there are a total of eighteen studies from the software engineering literature that contain human errors. Appendix C provides lists these eighteen papers.

2.4.2 Research question 2 – How can we organize the human errors identified in RQ1 into a taxonomy?

To create the human error taxonomy (HET), we followed two steps. First, each researcher examined the individual errors in Table 2.5 and grouped them into classes based on similarity. The two researchers met to discuss the classifications and agree upon a final list of error classes.

Then, each researcher individually organized the error classes into Reason's Slips, Lapses, Mistakes taxonomy (see Section 2.3 for more details on Reason's taxonomy). This organization entailed analyzing each error class to: decide whether the error class was an *execution error* or *an inadequate planning error*, and then, if the error class was an execution error, determine whether it was related to inattention failures (slips) or memory related failures (lapses).

The analyses mentioned above were performed by studying the error description or root cause description provided for the error by the primary study that supplied the error. The two researchers met and resolved all discrepancies and created an initial organization of the human error taxonomy.

Table 2.6. Human Error Taxonomy (HET)

Reason's Taxonomy	Human Error Class	Human Error(s) from Table 2.5
Slips	Clerical Errors	30, 31
	Term Substitution	5
Lapses	Loss of information from stakeholders	6
	Accidentally overlooking requirements	11
	Multiple terms for the same concept errors	5
Mstakes	Application Errors: knowledge-based plan is incomplete	1, 2, 18-25
	Solution Choice Errors	26, 27
	Syntax Errors	28, 29
	Wrong Assumptions: knowledge-based plan is wrong	3, 4, 14
	Environment Errors: knowledge-based plan is incomplete	15
	Information management Errors: knowledge-based plan is incomplete	16
	Inappropriate communication based on incomplete/faulty understanding of roles: knowledge-based plan is wrong	8
	Not having a clear distinction between client an users	9
	Mistaken belief that it is impossible to specify non-functional requirements: knowledge-based plan is incomplete	10
	Inadequate requirements process	13
	Lack of awareness of requirements sources: knowledge-based plan is incomplete	17
Violations	Assumption that insufficient requirements are ok	7
	Ignoring some requirements engineering tasks	12

Finally, human error expert Bradshaw evaluated and approved the final organization. Table 2.6 shows the final HET based on Reason's taxonomy. Notice that the last two rows are categorized as violations (i.e., deliberately failing to follow rules), which is different from

Reason's definition of unintentional errors (slips, lapses and mistakes). The human error taxonomy covers only unintentional human errors. Therefore, it excludes violations.

Next, to make the error classes in Table 2.6 more understandable, we provide a more detailed description of each one along with an example error and fault. For the examples, we use the Loan Arranger (LA) system, which was developed by researchers at the University of Maryland for use in software quality research. After a brief overview of LA, we present the error classes organized by the high-level groupings in Reason's taxonomy.

Loan Arranger (LA) overview: A loan consolidation organization purchases loans from banks and bundles them for resale to other investors. The LA application selects an optimal bundle of loans based on criteria provided by an investor. These criteria may include: 1) risk level, 2) principal involved, and 3) expected rate of return. The loan analyst can then modify the bundle as needed for the investor. LA automates information management activities, such as updating loan information provided monthly by banks.

2.4.2.1 Slips

A slip occurs when someone performs the steps required to complete a planned task incorrectly or in the wrong sequence. Slips generally result from a lack of attention. For example, the steps for "preparing breakfast cereal" include: (i) get a bowl, (ii) choose cereal, (iii) pour cereal in bowl, (iv) pour milk on cereal, and (v) eat cereal. If someone intends to 'prepare breakfast cereal', but pours orange juice rather than milk, that person has committed a slip. Table 2.7 lists the two types of slips that occur during the requirements phase.

2.4.2.2 Lapses

Table 2.7. Slip errors

Error Name	Description	Example of error	Example of fault
Clerical errors	Result from carelessness while performing mechanical transcriptions from one format or from one medium to another. Requirement examples include carelessness while documenting specifications from elicited user needs.	Error: In notes from the meeting with the client, the requirement author recorded a different set of parameters for jumbo loans and regular loans. When creating the requirements document, the author incorrectly copied the requirements for regular loans into the description of jumbo loans.	Fault: The requirements for the jumbo loans incorrectly specify the same behavior as for regular loans
Term substitution	Occur when requirement authors use the wrong term for a concept.	Error: The requirement author employs a common term incorrectly to refer to a more unusual concept.	Fault: Use of terms: "marked for inclusion" and "identified for inclusion" to convey the same information.

A lapse occurs when someone forgets the overall goal of the planned task in the middle of a sequence of actions or omits a step that is part of a routine sequence. Lapses generally result from memory-related failures. For example, someone enters a room to accomplish some specific task. If, upon entering the room, he forgets the planned task, he has committed a lapse. Table 2.8 lists the three types of lapses that occur during the requirements phase.

2.4.2.3 Mistakes

As opposed to slips and lapses, mistakes are errors in planning rather than execution, i.e., the plan is incorrect relative to the desired goal. In this scenario, someone executes the plan properly, but the plan itself is inadequate to achieve the intended outcome. For example, suppose that a taxi driver's dispatcher has informed him of a road closure on his typical route to the airport. If the taxi driver follows his typical route anyway and encounters a delay, the taxi driver

has made a mistake. The taxi driver devised an incorrected plan even though he had prior knowledge of the road closure. Table 2.9 lists the 11 mistake errors that occur during the requirements phase.

2.5. Discussion

This section provides further insights into the detailed results. Section 5.1 discusses overall findings about human errors in requirements engineering. Section 5.2 organizes the requirements errors by phases in the requirements engineering process. Section 5.3 analyzes the citations in the included papers to gain insight into the fragmentation of human error research in software engineering. Section 5.4 discusses the threats to validity of this review.

Table 2.8. Lapse errors

Error Name	Description	Example of error	Example of fault
Loss of information from stakeholders errors	Result from a requirement author forgetting, discarding or failing to store information or documents provided by stakeholders, e.g., some important user need	Error: A loan analyst informs the requirement author about the format for loan reports (file, screen, or printout), who forgets to note it down	Fault: Information about the report formats is omitted from the requirement specification.
Accidentally overlooking requirement errors	Occur when the stakeholders who are the source of requirements assume that some requirements are obvious and fail to verbalize them	Error: Because stakeholders assume that abnormal termination and system recovery is a commonplace occurrence and will be handled by the requirement analysts or the system design team, they do not provide system recovery requirements.	Fault: Requirements document does not describe the process of system recovery from abnormal termination.
Multiple terms for the same concept errors	Occur when requirement authors fail to realize they have already defined a term for a concept and so introduce a new term at a later time.	Error: The same concept is referred to by different terms throughout the document.	Fault: Use of terms: “marked for inclusion” and “identified for inclusion” to convey the same information.

2.5.1 Findings about human errors in requirements engineering

After analyzing the included papers, identifying the presence of human errors, and organizing those errors into a taxonomy, we can report some overall findings about human errors in requirements engineering.

Table 2.9. Mistake errors

Error Name	Description	Example of error	Example of fault
Application errors: knowledge-based plan is incomplete.	Arise from a misunderstanding of the application or problem domain or a misunderstanding of some aspect of overall system functionality	Error: The requirements author lacks domain knowledge about loans, investing, and borrowing. As a result, she incorrectly believes that the stakeholders have provided all information necessary to decide when to remove loans that are in a default status from the repository.	Fault: The requirements specification omits the requirement to retain information about borrowers who are in default status (even after the corresponding loans are deleted from the system).
Environment errors: knowledge-based plan is incomplete.	Result from lack of knowledge about the available infrastructure (e.g., tools, templates) that supports the elicitation, understanding, or documentation of software requirements.	Error: The requirement authors failed to employ a standard template for documenting requirements (for example, the IEEE standard template for SRS) because they were unaware of the presence of such a template.	Fault: Without the support of the template, requirements about system scope and performance were omitted.
Solution Choice errors	These errors occur in the process of finding a solution for a stated and well-understood problem. If RE analysts do not understand the correct use of problem-solving	Error: Lack of knowledge of the requirement engineering process and requirement engineering terminology on part of	Fault: A particular requirement listed under performance requirement should be a functional requirement.

	methods and techniques, they might end up analyzing the problem incorrectly, and choose the wrong solution	the analysts. The analyst cannot differentiate between performance and functional requirements.	
Syntax errors	Occur when a requirement author misunderstands the grammatical rules of natural language or the rules, symbols, or standards in a formal specification language like UML.	Error: The requirements engineer misunderstood the use of navigability arrows to illustrate that one use case extends another.	Fault: An association link between two classes on a UML diagram lacks a navigability arrow to indicate the directionality of association resulting a diagram that is ambiguous and can be misunderstood.
Information Management errors: knowledge-based plan is incomplete.	Result from a lack of knowledge about standard requirement engineering or documentation practices and procedures within the organization	Error: The organization has a policy that requirement specifications include error-handling information. Unfamiliar with this policy, the requirements engineer fails to specify error-handling information.	Fault: Specification does not indicate that an error message should be displayed regarding errors rather than just returning to a previous screen with no indication or notification of the problem.
Wrong Assumption errors: knowledge-based plan is wrong	Occur when the requirements author has a mistaken assumption about system features or stakeholder opinions	Error: Requirements author assumes that error-handling is a task common to all software projects and will be handled by programmers. Therefore, s/he does not gather that information from stakeholders.	Fault: Information about what happens when a lender provides invalid data has been omitted.
Mistaken belief that it is impossible to specify non-functional requirements: knowledge-	The requirements engineer(s) may believe that non-functional requirements cannot be captured and therefore omit this process from their elicitation and development plans.	Error: The requirements engineering team failed to consider non-functional requirements.	Fault: Omission of performance requirements, security requirements and other non-functional requirements.

based plan is incomplete			
Not having a clear distinction between client and users	If requirement engineering practitioners fail to distinguish between clients and end users, or do not realize that the clients are distinct from end users, they may fail to gather and analyze the end users' requirements	Error: The requirement-gathering person failed to gather information from the actual end user of LA system, the Loan Analyst.	Fault: No functional requirement to edit loan information has been specified, even though the 'Purpose' specifies loans can be edited.
Lack of awareness of requirement sources: knowledge-based plan is incomplete	Requirements gathering person is not aware of all stakeholders which he/she should contact in order to gather the complete set of user needs (including end users, customers, clients, and decision-makers).	Error: Requirement gathering person was not aware of all end users and clients and did not gather the needs of a bank lender (one of the end users of LA system). This end user wanted the LA system to handle both fixed rate loans and adjustable rate loans.	Fault: Omitted functionality as requirements only considers fixed rate loans.
Inappropriate communicatio n based on incomplete/fa ulty understanding of roles: knowledge-based plan is wrong	Without proper understanding of developer roles, communication gaps may arise, either by failing to communicate at all or by ineffective communication. The management structure of project team resources is lacking.	Error: Team members did not know who was supposed to elicit the requirements from a bank lender, which affected the participation of an important stakeholder during the requirements process.	Fault: Omitted functionality as requirements of a bank lender (i.e., the LA application system to handle both fixed rate loans and adjustable rate loans) were not recorded in the specification.
Inadequate Requirements Process	Occur when the requirement authors do not fully understand all of the requirement engineering steps necessary to ensure the software is complete and neglect to incorporate one or more essential steps into the plan.	Error: Requirement engineering plan did not incorporate requirement traceability measures to link requirements to user needs.	Fault: An extraneous requirement that allows loan analysts to change borrower information is included that could result in unwanted functionality and unnecessary work for the developers.

- ***Difficulty of identifying human errors in requirements engineering***. The term *error* is used inconsistently in the requirements engineering literature. Even when applied to human thoughts or actions, it often did not match the definition of *human error* from psychology. To ensure that we retained only human errors, we had to examine each reported *error* in detail to remove those that discussed different types of problems. Through this process we removed a considerable majority of the items marked as *errors* in the literature (see Section 2.4.1), indicating inconsistent use of the term *error*. Only 18 papers reported a true *human error*. The broader set of problems that researchers called *errors*, while outside the scope of this review, are worthy of further attention.
- ***The need for closer integration of software engineering and psychology research***. The development of the human error taxonomy required close collaboration between software engineering experts and a human error expert from psychology. While human error researchers have developed a number of general frameworks for describing and classifying human errors, many of the errors found in the software engineering literature did not use these frameworks. As a result, we had to carefully analyze each error description to determine whether it truly was a human error. Our work has shown that a close collaboration between software engineering researchers and psychology researchers can help provide requirements engineering errors with a theoretically-based *human error* framework for organizing the error information.
- ***Errors vs. Violations***. It is also important to note that the human error literature distinguishes between unintentional errors and deliberate violations. This distinction is important because the two problems need to be handled in very different ways. Some 'error' reports were, in fact, violations, such as a situation where software engineers

deliberately omitted some tasks in the requirements engineering process. Preserving the distinction between errors and violations is useful because detection and remediation techniques differ for the two types of problem. Violations, although infrequent in the target articles we identified in our systematic literature review, fall outside the scope of the current systematic literature review.

2.5.2 Structuring errors within specific requirements engineering tasks

The requirements engineering process contains a series of phases, each of which engenders opportunities for human error. Because the types of activities that occur in each phase differ, so might the errors that occur. To help developers understand how the errors can be distributed across these phases, we developed a second taxonomic organization of errors that arranges errors by both the activity where they occur and by the slips/lapses/mistakes distinction, as shown in Table 2.10.

There are five distinct activities in requirements engineering:

- *Elicitation*: Discovering and understanding the user's needs and constraints for the system.
- *Analysis*: Refining the user's needs and constraints.
- *Specification*: Documenting the user needs and constraints clearly and precisely.
- *Verification*: Ensuring that the system requirements are complete, correct, consistent, and clear.
- *Management*: Scheduling, coordinating, and documenting the requirements engineering activities.

While each of the three error types can occur during each requirement activity, the specific error classes will differ. Two researchers each mapped the error classes onto the

requirements activities based on the knowledge of the requirements engineering activities and the error description. As before, the two authors discussed and resolved any discrepancies regarding the mapping of error classes across the requirements activities. Human error expert Bradshaw reviewed and approved the final mapping shown in Table 2.10. Note that many of the error classes appear across multiple requirements activities. For example, a clerical error can occur:

- During *elicitation* if the requirements gathering person is careless while taking notes during the interview, or
- During *specification* if the requirements author is careless while transcribing the elicited requirements into specifications.

As before, the two authors discussed and resolved any discrepancies regarding the mapping of error classes across the requirements activities. Human error expert Bradshaw reviewed and approved the final mapping.

There are several reasons why we believe this two-dimensional structure in Table 2.10) is superior to the one-dimensional structure introduced in Table 2.6. First, the two-dimensional structure allows us to observe that the same error class may appear in different activities, which we could not observe from the one-dimensional structure. Second, we anticipate that as investigators identify new errors, those errors will populate additional cells in the table. The paucity of existing error reports, coupled with the prevalence of faults and failures in software, suggests that many errors are occurring that have not been recognized formally. The two-dimensional table helps to *predict* specific errors that have yet to be identified.

An analogy may help explain our thinking. As Mendeleev was working on the Periodic Table of Elements, only about 60 elements had been discovered. His table had ‘holes’ where elements of a particular atomic weight and valence should exist. Indeed, the absence of these

elements was considered problematic for the Periodic Table. Over the next 15 years, chemists discovered three of the missing elements. Those elements had the properties predicted by the Periodic Table (Scerri 2006). Similarly, we can predict that **clerical errors** will be observed, not only in *elicitation* and *specification,* but also in the other activities. Investigators can employ this table to help identify errors in different requirements engineering activities and 'find the missing elements' in our table.

Table 2.10. Mapping human errors to requirements engineering activities

RE Activities / Human Error Categories and Classes		Elicitation	Analysis	Specification	Verification	Management
Slips	Clerical Errors	✓		✓		
	Term Substitution			✓		
Lapses	Loss of information from stakeholders	✓				
	Accidentally overlooking requirements	✓				
	Multiple terms for the same concept errors			✓		
Mistakes	Application errors: knowledge-based plan is incomplete.	✓	✓		✓	
	Environment errors: knowledge-based plan is incomplete.	✓	✓	✓	✓	
	Information Management errors: knowledge-based plan is incomplete.					✓
	Wrong Assumption errors: knowledge-based plan is wrong	✓	✓			
	Inappropriate communication based on incomplete/faulty understanding of roles: knowledge-based plan is wrong	✓	✓		✓	
	Mistaken belief that it is impossible to specify non-functional requirements: knowledge-based plan is incomplete	✓	✓		✓	
	Not having a clear demarcation between client and users	✓				
	Lack of awareness of requirement sources: knowledge-based plan is incomplete	✓				

	Solution choice errors		✓			
	Inadequate Requirements Process					✓
	Syntax errors			✓		

2.5.3 Usefulness of Human Error Taxonomy

The current paper reports the results on collection and classification of human errors across various requirements engineering (RE) activities. Since development of the HET, we have empirically validated the usefulness of error detection, fault detection, error prevention, and training of software engineers on retrospective human error analysis.

From a defect detection perspective, a deeper understanding of the human cognitive failures that affected the development of a particular requirements artifact can lead developers to detect additional defects that are often overlooked when the focus is on faults (i.e., manifestations of human error). We conducted feasibility studies (Anu et al., 2016; Hu et al., 2016) to ensure that the developers can use the structured human error information in the HET to improve their defect detection ability during a requirements inspection. The result of the studies showed use of human error information allowed participants to identify a significantly large number of faults that they missed when using a fault-checklist inspection approach.

We also anticipate that the HET can help developers better understand the human errors that occur most frequently during the requirements development process. This knowledge is likely to help developers avoid these human errors, and their related faults, when developing requirements artifacts. We conducted an empirical study to test the effect of knowledge of human errors on error and fault prevention (Hu et al., 2017). The results of this study confirmed that a better understanding of human errors led participants to insert fewer errors and faults into their requirements artifacts.

More recently, we have used the HET to train inspectors to retrospectively analyze faults to determine the underlying human error committed during requirements development. This process is similar to that of incident/accident investigation done in Aviation and Medicine domains. Under the advisement of our Psychology expert (also a co-author), a training instrument has been developed (available at http://vaibhavanu.com/NDSU-CS-TP-2016-001.html) that can be used to train inspectors to investigate requirements faults (Anu et al., 2017) using an error-abstraction decision tree framework called Human Error Abstraction Assist (HEAA).

2.5.4 Citation analysis to analyze the fragmentation in human error research

Our observation, that among the original 38 software engineering papers there were 36 publication venues represented, triggered an exploratory citation analysis to understand the coherency, or lack thereof, of human error research in software engineering. In this more detailed analysis, we used only the 18 papers from the literature search that contain true human errors. We extracted the reference lists from these papers. We then compared those reference lists to identify cases where one paper cited another. Because our search was thorough, we have high confidence that this set of 18 papers includes all of the papers in the software engineering literature that describe human errors in the requirements phase. In addition, we identified citations to any papers on human error from psychology.

In the results shown in Figure. 2.2, an arrow from one paper to another indicates that the first paper cited the second paper. The result shows that only seven papers cited human error papers (either in software engineering or in psychology), implying that researchers may be unaware of related research. Of those papers, six made eight citations to previous work from the software engineering literature, and three included six citations to the psychological literature on errors. Four papers were cited by others but did not themselves cite related papers. Seven papers

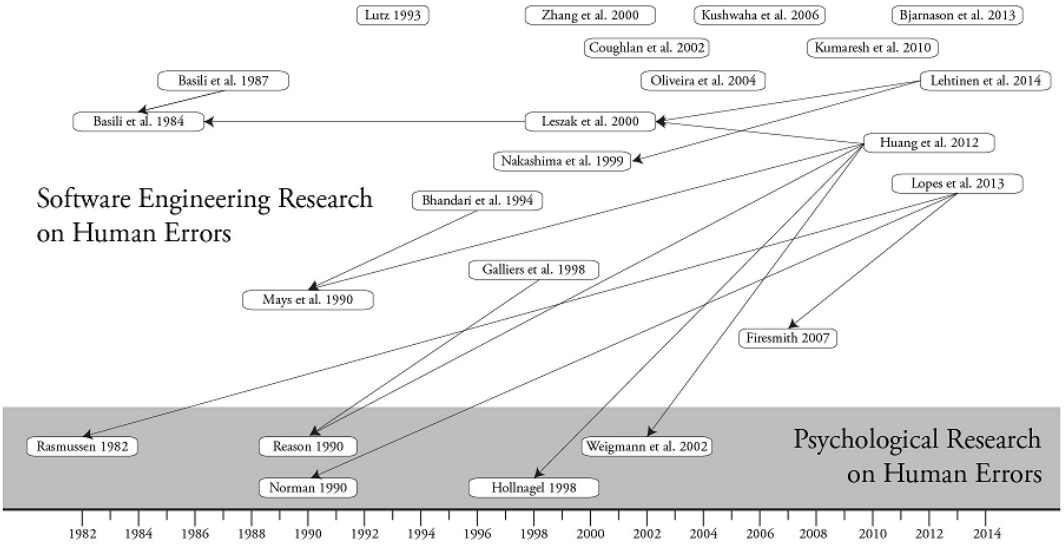


Figure 2.2. Citation analysis

were 'island papers,' citing none of the previous research and not receiving any citations from other papers. Two papers (Huang et al., 2012; Lopes and Forster 2013) were responsible for the majority of the identified citations.

Overall, Figure 2.2 illustrates that human error research in software engineering is in a formative state. Many investigators appear to be unaware of related work in the field because they produce independent papers that do not build on others' contributions. Furthermore, researchers are not building on human error research published in the psychological literature. It is likely that the wide distribution of papers across different research venues contributes to this problem. Researchers do not have a small number of venues from which to draw sources. Finally, a subtle, but encouraging trend emerges with more recent papers. Those papers, especially the ones published from 2012 onwards have more citations to earlier work. This trend suggests a growing awareness of prior work in the area.

2.5.5 Threats to validity

Threats addressed: To gather evidence across multiple avenues, we searched multiple electronic databases covering different software engineering and psychology journals and conferences. To reduce the *researcher bias*, two researchers independently performed each step during the review execution (i.e., search string, filtering results, data extraction and selection of primary studies) before consolidating the results. This process also ensured that the researchers extracted consistent information from each primary study. We performed both the identification of requirements phase human errors and the selection of a suitable human error classification system under the supervision of human error expert Bradshaw. This supervision ensured that the HET would be strongly grounded in Cognitive Psychology research.

Threats unaddressed: We conducted this review based on the assumption that SLR-2009 (Walia and Carver 2009) included all relevant studies prior to 2006. We are aware that such an assumption may have resulted in missing some studies and consequently missing some requirements phase human errors.

2.6. Conclusion and future work

A careful application of human error research to requirements engineering resulted in the development of a sound, theoretically sound taxonomy of the types of human errors that requirements engineers can commit. This taxonomy is based on a standard human error taxonomy from cognitive psychology. The primary contributions of this work are:

- Illustration of the ability and value of applying human error research to a software engineering problem through close interaction between software engineering experts and cognitive psychology experts;
- Development of a human error taxonomy to organize the common human errors made during requirements into a systematic framework; and

- An analysis of the literature describing human errors to identify the lack of cohesiveness in software engineering research as it relates to the use of human error information.

This study of human errors has the potential to provide a more effective solution to the software quality problem than a focus only on faults. A taxonomy like the one developed in this paper will help developers identify the most common types of errors so they can either implement policies to prevent those errors (e.g., training) or focus the review process on identification and removal of the faults caused by those errors.

Although the primary goal of this review was to propose a taxonomy of human errors during the requirements phase of software development, this review may be helpful for other researchers who want to create human error taxonomies for other phases of the software development lifecycle. While we have empirically validated the usefulness of HET for detecting and preventing requirements errors through a series of controlled studies, we expect to gather additional evidence and observations on additional requirements errors (see missing fields in Table 2.10) through empirical evaluations.

2.7. Acknowledgements

This research was supported by National Science Foundation Awards 1423279 and 1421006. The authors would like to thank the participants that attended the Workshop on Applications of Human Error Research to Improve Software Engineering (WAHESE 2015).

References

[1] Anu V, Walia GS, Hu W, Carver JC, Bradshaw G (2016) Effectiveness of Human Error Taxonomy during Requirements Inspection: An Empirical Investigation. In: Proc. 28th Int. Conf. Softw. Eng. Knowl. Eng. SEKE 2016. San Francisco, CA, USA, pp 531–536

[2] Basili V, Green S, Laitenberger O, Lanubile F, Shull F, Sørumgard S, Zelkowitz MV. (1996) The Empirical Investigation of Perspective-Based Reading. Empir Softw Eng 1:133–164. doi: 10.1007/BF00368702

[3] Bjarnason E, Wnuk K, Regnell B (2011) Requirements are slipping through the gaps - A case study on causes & effects of communication gaps in large-scale software development. In: Proc. 2011 IEEE 19th Int. Requir. Eng. Conf. RE 2011. pp 37–46

[4] Chillarege R, Bhandari IS, Chaar JK, Halliday MJ, Ray BK, Moebus DS (1992) Orthogonal Defect ClassificationA Concept for In-Process Measurements. IEEE Trans Softw Eng 18:943–956. doi: 10.1109/32.177364

[5] Diller T, Helmrich G, Dunning S, Cox S, Buchanan A, Shappell S (2014) The Human Factors Analysis Classification System (HFACS) applied to health care. Am J Med Qual 29:181–90. doi: 10.1177/1062860613491623

[6] Firesmith D (2007) Common requirements problems, their negative consequences, and the industry best practices to help solve them. J Object Technol 6:17–33. doi: 10.5381/jot.2007.6.1.c2

[7] Glaser BG, Strauss AL (1967) The discovery of grounded theory. Int J Qual Methods 5:1–10. doi: 10.2307/588533

[8] Hu W, Carver JC, Anu V, Walia GS, Bradshaw G (2016) Detection of Requirement Errors and Faults via a Human Error Taxonomy: A Feasibility Study. In: Proc. 10th ACM/IEEE Int. Symp. Empir. Softw. Eng. Meas.

[9] Hu W, Carver JC, Anu V, Walia GS, Bradshaw G (2017) Defect Prevention in Requirements using Human Error Information: An Empirical Study. In: 23rd International Working Conference on Requirements Engineering: Foundation for Software Quality.

[10] Huang F, Liu B, Huang B (2012) A Taxonomy System to Identify Human Error Causes for Software Defects. Proc. 18th Int. Conf. Reliab. Qual. Des. ISSAT 2012

[11] IEEE (2010) Systems and software engineering -- Vocabulary. ISO/IEC/IEEE 24765:2010(E) 1–418. doi: 10.1109/IEEESTD.2010.5733835

[12] Kitchenham B (2004) Procedures for performing systematic reviews. Keele, UK, Keele Univ 33:28. doi: 10.1.1.122.3308

[13] Kumaresh S, Baskaran R (2010) Defect Analysis and Prevention for Software Process Quality Improvement. Int J Comput Appl 8:42–47. doi: 10.5120/1218-1759

[14] Kushwaha DS, Misra AK (2006) Cognitive software development process and associated metrics - A framework. In: Proc. 5th IEEE Int. Conf. Cogn. Informatics, ICCI 2006. pp 255–260

[15] Lanubile F, Shull F, Basili VR (1998) Experimenting with Error Abstraction in Requirements Documents. Proc. 5th Int. Symp. Softw. metrics

[16] Lehtinen TOA, Mantyla MV., Vanhanen J, Itkonen J, Lassenius C (2014) Perceived causes of software project failures - An analysis of their relationships. Inf Softw Technol 56:623–

643. doi: 10.1016/j.infsof.2014.01.015

[17] Leszak M, Perry DE, Stoll D (2000) A case study in root cause defect analysis. In: Proc. 22nd Int. Conf. Softw. Eng. ICSE 2000. Limerick, Ireland, pp 428–437

[18] Lopes M, Forster C (2013) Application of human error theories for the process improvement of Requirements Engineering. Inf Sci (Ny) 250:142–161.

[19] Mays R, Jones C, Holloway G, Studinski D (1990) Experiences with defect prevention, IBM Systems Journal 29 (1): 4–32

[20] Reason J (1990) Human error. Cambridge University Press, New York, NY

[21] Reason J (2008) The human contribution: unsafe acts, accidents and heroic recoveries. Ashgate Publishing, Ltd., Surrey, UK

[22] Reason J, Manstead A, Stradling S, Baxter J, Campbell K (1990) Errors and violations on the roads: a real distinction? Ergonomics 33:1315–1332. doi: 10.1080/00140139008925335

[23] Scerri ER (2006) The Periodic Table: Its Story and Its Significance. Oxford University Press, USA.

[24] Scherer D, Maria de Fátima Q V., Neto JADN (2010) Human Error Categorization An Extension to Classical Proposals Applied to Electrical Systems Operations. Human-Computer Interact 234–245.

[25] Tassey G (2002) The Economic Impacts of Inadequate Infrastructure for Software Testing. Gaithersburg, MD

[26] Walia GS, Carver JC (2009) A systematic literature review to identify and classify software requirement errors. Inf Softw Technol 51:1087–1109. doi: 10.1016/j.infsof.2009.01.004

[27] Walia GS, Carver JC (2013) Using error abstraction and classification to improve requirement quality: Conclusions from a family of four empirical studies. Empir Softw Eng 18:625–658. doi: 10.1007/s10664-012-9202-3

[28] Wiegmann D, Detwiler C (2005) Human Error and General Aviation Accidents : A Comprehensive, Fine-Grained Analysis Using HFACS. Security 1–5. doi: 10.1518/001872007X312469

[29] Zhang J, Patel VL, Johnson TR, Shortliffe EH (2004) A cognitive taxonomy of medical errors. J Biomed Inform 37:193–204. doi: 10.1016/j.jbi.2004.04.004

CHAPTER 3

EVALUATING THE USEFULNESS OF HUMAN ERROR TAXONOMY THROUGH TWO QUASI-CONTROL STUDIES

3.1. Introduction

The software development process is human-centric as developers interact with customers and users to document customer needs into software requirements and then transform those requirements into architecture, design, and ultimately code. Much research and experience has shown that developers introduce problems during various stages of the development process. Identifying and correcting such problems can be quite costly. Software engineering researchers have devoted significant attention to improving software quality. Most of this attention has traditionally focused on approaches that address the faults, i.e., the problems recorded in various software engineering artifacts. To provide a comprehensive picture of the problem, it is also important to understand and address the underlying cause of the faults, i.e., the human mental errors. Identifying these human mental errors can help developers understand why problems occurred, find and fix related faults, and prevent errors from happening in the future.

The development of the software requirements specification (SRS) is the first and most crucial stage in the software engineering lifecycle. During this phase, developers communicate with various stakeholders to elicit and document customer needs and specify the solution. Development of the SRS is especially defect-prone because of the ambiguities of natural language and the concomitant difficulty in communicating with all relevant stakeholders. If these SRS defects are not detected and repaired at the requirements stage, they become more

expensive to fix. Developers often spend 40-50% of their effort fixing problems that should have been fixed earlier in the software lifecycle [1]. Therefore, our work focuses on identifying the human mental errors that caused the requirements defects to reduce the chances of those defects propagating to later stages.

Other researchers have recognized this need to understand the human errors that cause the faults, and have proposed various solutions. Root Cause Analysis (RCA) helps developers identify systematic development problems, characterize the source of faults, and identify process improvement needs based on those sources [9]. Orthogonal Defect Classification (ODC) provides in-process feedback to developers and helps them identify faults as early as possible [3]. Error Abstraction helps developers identify the errors that underlie the faults detected during an inspection and provides guidance on how to reinspect to find additional errors and faults [7]. While all of these approaches seek to understand the source of problems, none of these have a strong connection to work in cognitive psychology, which has a long history of studying human errors and their manifestations and consequences in many domains.

Because the software development process is a human-based activity, it is reasonable to investigate the application of insights from research in cognitive psychology. Requirements faults are often the result of failings in human thought. For example, a case study by Trevor and Jim found that software problems resulted from phenomena associated with human mental processes, which were not directly related to software engineering [14]. In another example, Lopes and Forster provided a proof of concept that the use of human error theories can improve the quality of software requirements [10]. These studies help establish that the use of human error theories can improve software quality by helping developers better understand problems in the development of the SRS and by reducing the number of inserted defects.

Employing error research from the cognitive psychology literature, we developed a taxonomy of human errors that can occur during the development of an SRS. Our Human Error Taxonomy (HET) was informed by a systematic literature review that combines research from software requirements with research from cognitive psychology. Working with a Cognitive Psychologist, we analyzed the human error literature to determine how cognitive theories can be applied to software engineering problems. As a result, our taxonomy is theoretically-grounded in appropriate research results. Section 3.2.3 provides an overview of the HET.

The goal of this chapter is to investigate the feasibility and description of the HET to support the SRS inspection process and represent the real requirements problems. The primary contributions of this article are (1) evaluation of the feasibility of the HET for supporting inspection of an SRS, (2) an analysis of the types of human errors that novice developers make while developing an SRS, and (3) evaluation the description of HET for real industry problems.

The remainder of this paper is organized as follows: Section 3.2 describes the proposed approach. Section 3.3 provides a description of a feasibility study conducted to evaluate the effectiveness and efficiency of the proposed approach. Section 3.4 describes the analysis and results of the feasibility study. Section 3.5 summarizes and discusses the results. Section 3.6 provides a description of an interview and survey study conducted to evaluate the representation of HET for real industry problem. Section 3.7 discusses the comparison of these two studies, Section 3.8 discusses the validation threat of these two studies, followed by a brief conclusion of this paper and ideas for future studies in Section 3.8.

3.2. Research method

This section provides background information on topics relevant to our study. Section 3.2.1 describes previous research showing the value of building taxonomies of software

requirements problems. Section 3.2.2 explains the process of error abstraction. Section 3.2.3 describes the HET that is the subject of this study.

3.2.1. Fault-Based Taxonomies

Because this work makes use of a taxonomy of problems, it is important to understand prior work on creating taxonomies of software problems. Researchers studying early-lifecycle quality improvement methods have developed taxonomies of software faults. Subsequent empirical evaluation showed that such taxonomies can help developers improve software quality [1-4]. However, these approaches cannot help developers identify the full spectrum of faults or how to improve development practices based on such faults [5]. These taxonomies are also incomplete [6]. Thus, there is a need for new approaches that extend existing taxonomies of problems, but overcome their shortcomings. Specifically, there is a need to focus on the sources of the problems (i.e., the human error).

3.2.2. Error Abstraction Process

The error abstraction process used in this paper is based on previous work by Lanubile et al. [7]. Error abstraction helps developers identify the root causes of requirements faults. Each inspector will be trained on how to abstract the underlying errors from related faults. The relationship between errors and faults is many-to-many, which is one error may cause multiple faults and one fault may be caused by multiple errors. For example, the error of lack of communication between end users and requirements engineers can lead to the faults of missing information or incorrect information. Thus, using the error abstraction will help inspectors understand the underlying causes for related faults and help them identify additional faults, related to the abstract error, in the SRS.

3.2.3. Human Error Taxonomy (HET)

Previously, Walia and Carver developed an initial Requirement Error Taxonomy (RET) to provide structure to the process of identifying human errors in the development of SRS documents [8]. They subsequently validated the usefulness of the RET through a series of empirical studies [9]. Key findings from these studies include (1) use of structured error information improves fault detection effectiveness compared with using only fault information, and (2) the use of human error information is beneficial for the prevention of requirements problems. The primary weakness of the RET was that, while it was inspired by human error research in cognitive psychology, it lacked a strong and direct tie to existing human error taxonomies. To address this weakness, we developed the HET.

In order to build a formal and theoretically sound error taxonomy, we conducted a systematic literature review. First, we analyzed how the findings from human error research can be applied to the software engineering domain. Then, we analyzed human error research published in the software engineering literature to determine the degree of fit to psychological literature on human errors in cognitive psychology research. The results of this analysis was a set of 15 error classes that described common problems that occur during the development of SRS documents. The last step was to map these error classes into one of the existing human error taxonomies from cognitive psychology.

Then, we mapped information about requirement error from the software engineering literature to human error taxonomies from the cognitive literature. After examining a number of taxonomies, we chose the *Slips*, *Lapses*, and *Mistakes* taxonomy by Reason [10]. We chose this taxonomy because it establishes a strong association between the human information processing model and cognitive mechanisms like inattention, memory failures, and lack of knowledge.

These cognitive mechanisms are easy to understand and relate to when classifying a given human error into the category of slip, lapse or mistake.

Slips and *Lapses* describe errors that occur when a planned action was not executed or completed as intended. A *Slips* error occurs when someone carries out a planned task incorrectly or in the wrong sequence. For example, someone intended to pour milk on his/her morning cereal but instead pours orange juice instead.

Lapses errors are generally memory related failures. They occur when someone forgets a goal in the middle of a sequence of actions or omits a step in a routine sequence. For instance, someone enters a room to accomplish some task, but upon entering the room forgets the planned task. Conversely a *Mistake* is a planning error in that someone designed an incorrect plan to achieve the desired goal. In this error, the plan is executed properly, but the plan itself was inadequate to achieve the intended outcome. For example, suppose you are driving your usual route work and encounter traffic. You devise an alternate plan to travel to work via a different route. On your new route, you encounter unexpected road construction and must return to the original route. Your plan was wrong because your model of the city traffic pattern was wrong. Full details of the HET and its development are beyond the scope of this paper. Figure 2.2 in Chapter 2 provides an overview of the HET.

One example of an error from the *Environment* class of the *Mistake* type in the HET and the resulting fault is:

Error: When documenting software requirements, the developers did not use a standard requirements specification template (e.g., the IEEE standard). In this case, the correct tools for requirements specification were not used.

Fault: Requirements about the system scope and performance were omitted.

3.3. Experiment design of feasibility study

The major goal of this study is *to investigate the feasibility of using the HET to support the SRS inspection process*. The following subsections describe the research questions and hypotheses, the participants, and study procedures.

3.3.1 Research Questions and Hypotheses

To address the overall study goal, we explored three distinct research questions. Along with each research question, we define specific research hypotheses.

RQ1: Does the HET provide a useful method of describing and classifying the errors and faults made during development of an SRS?

As described in Chapter 2, the HET is comprised of three high-level error types and fifteen detailed error classes. Based on our literature review, each of these error types and classes has both theoretical and empirical support. By incorporating previous reports of errors made during the development of an SRS, the error classes in the taxonomy are likely to re-occur at a later time in the same SRS development effort or in future SRS development efforts. The focus of this research question is to understand whether these error types and classes actually describe real problems that occur in the SRS. To ensure that the taxonomy is valid, we also need to understand whether all of the error types and classes are necessary.

Hypothesis 1: Developers will find errors and related faults that fit into each of the three high-level error types and the fifteen low-level error classes (i.e., the observed frequency will be greater than 0).

Based on our HET, the *Mistakes* error type has the largest number of error classes. The study of Kraemer and Carayon also pointed out that the mistake errors are the most common in the area of computer and information security [11].

Hypothesis 2: Developers will make more *Mistake* errors and faults.

RQ2: Do the errors identified by the HET lead to additional faults being identified during reinspection?

Once a development team becomes aware that they have made an error during the creation of the SRS, they may identify additional faults related to that error. To further understand the usefulness of the HET, we need to determine whether knowledge of errors can lead developers to find additional related faults.

Hypothesis 3: Development teams will find additional faults when reinspecting their SRS document guided by knowledge of existing errors (i.e., the observed frequency will be greater than 0).

RQ3: Do developers think the HET is helpful?

This question focuses on the developer subjective impression of the HET. Similar to the prior work of Walia and Carver [9], because error abstraction and classification is a subjective process, we also measured effectiveness through a developer survey after using the HET.

Hypothesis 4: Developers believe that the HET is useful for abstracting and classifying requirement errors.

3.3.2 Participants

The study included twenty-eight senior-level undergraduate computer science students enrolled in the Fall 2015 capstone project course at The University of Alabama. The primary course goal was for student teams to proceed through requirements elicitation/documentation, design, implementation, and testing to build a complete software system. Separate from the design of this study, the course instructor (who is not part of the research team) divided the students into eight three- or four-person teams. Each team developed their own system (as described in Table 3.1).

3.3.3 Procedure

The study procedure included two training sessions, five experimental steps, and a post-study survey. Figure 3.1 illustrates the process using notation shown in Figure 3.2.

Table 3.1. Teams and System Description

Team #	System Name and Description
1	**Grade Keeper**: manage student course information
2	**Tenant Tracker**: provide communication between property managers and tenants
3	**Nutrition**: keep track of the food the users have eaten
4	**Opus**: share pictures of interesting items in the local community
5	**Robotics Score**: manage the scores for a robotics competition
6	**Promo Pass**: manage business promotion information for customers
7	**Bowling**: create and manage a bowling game session
8	**Is Open**: manage detailed information about local businesses

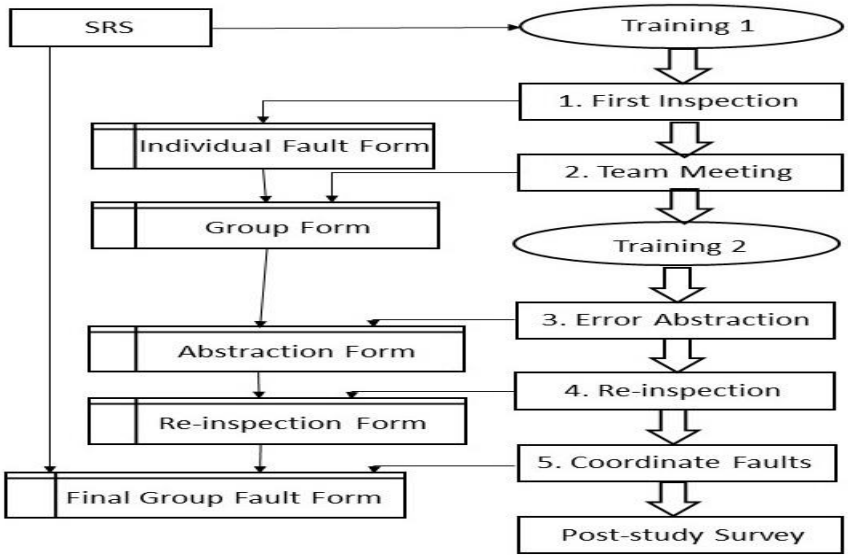


Figure 3.1. Experiment Procedure

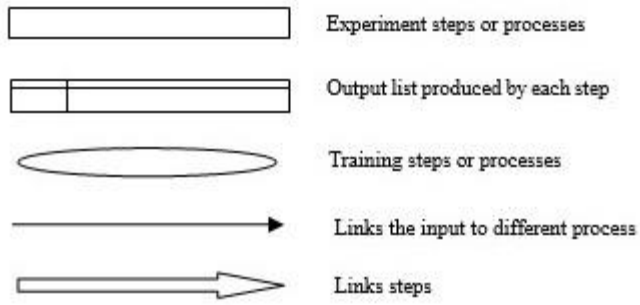


Figure 3.2. Experiment Notation

Training 1 - Fault checklist technique: The goal of the initial training was to introduce the types of faults that can occur when developing an SRS. Using a requirements fault checklist [13], we provided detailed information about the important types of faults. We also trained the participants on how to use the checklist to inspect an SRS and on how to record the faults found during that inspection. This training session lasted approximately 15 minutes.

Step 1- First Inspection (Individual Inspection): We randomly assigned each participant to evaluate an SRS developed by another team. The participants then used the fault checklist (from Training 1) to inspect the assigned SRS and identify faults. Each participant completed an *Individual Fault Form* to list the faults found during the inspection. The output of this step was 28 *Individual Fault Forms* (one per participant).

Step 2 - Team Meeting to Consolidate Faults: Each team received the *Individual Fault Forms* submitted by the participants who inspected their SRS (Step 1). As a group, each team examined these fault lists to remove duplicates and consolidate the faults into one master list, which they recorded on the *Group Form*. The output of this step was eight *Group Forms* (one per team).

Training 2 - Error abstraction and classification: During the first 20 minutes, we trained the participants on the error abstraction process, including how to abstract errors from faults. Then we spent 20 minutes training the participants on the HET and how to use it to abstract and classify requirements errors. We described each HET error class and provided an illustrative and detailed example.

Step 3 - Error Abstraction: Based on Training 2, each team used the HET to abstract and classify the errors recorded on their *Group Form* from Step 2. They recorded these errors in an *Abstraction Form*. The output of this step was eight *Abstraction Forms* (one per team).

Step 4 - Re-inspecting the SRS: Using the errors abstracted in Step 3, each participant individually reinspected their own SRS document to identify any additional faults related to these errors (i.e., faults not found by their classmates in Step 1). The participants recorded these additional faults on the *Reinspection Form*. The output of this step was 28 *Reinspection Forms* (one per participant).

Step 5 - Coordinating the individual fault lists: This step is equivalent to Step 2, except that the teams used the faults found in the reinspection (Step 4). Each team produced a *Final Group Fault Form* reporting the agreed-upon list of faults. The output of this step was eight *Final Group Fault Form* (one per team).

Post-study Survey: At the end of the study, each participant provided subjective feedback about the error abstraction process and the use of the HET. The details of the survey and the specific questions are listed in Table 3.5 in Section 3.5.3.

3.3.4 Data Cleaning and Analysis Process

After completing the study procedure, we determined that Teams 7 and 8 did not follow steps 2-5 correctly and failed to provide the required data. The members of these teams did perform Step 1 correctly to provide input to the other teams. Therefore, for the remainder of the

chapter, we report the results for the six teams that followed the instructions for steps 2-5. These six teams contained a total of twenty-one participants. For the statistical analysis, we selected an alpha value of 0.05 to judge the statistical significance of the results.

3.4. Analysis and result

This section provides a detailed analysis of the data collected during this study. The analysis is organized around the hypotheses posed in Section 3.2.1

3.4.1 RQ1: Does the HET provide a useful method of describing and classifying the errors and faults made during development of an SRS?

Because we are interested in characterizing the overall usefulness of the HET, for the analysis in this section, we include data from the entire process (i.e., the *Individual Fault Forms, Group Forms, Abstraction Forms, Reinspection Forms, and Final Group Fault Forms*).

H1: Developers will find errors and related faults that fit into each of the three high-level error types and the fifteen low-level error classes (i.e., the observed frequency will be greater than 0).

First, for the high-level error types, Figure 3.3 shows that all six teams made errors from each of the three error types. We ran three one-sample t-tests to determine whether the frequency errors of each type was significantly greater than 0. In all three cases, the number of errors was significantly higher than 0: *Slips* (p=.01), *Lapses* (p=.005), and *Mistakes* (p=.001).

Figure 3.4 shows the distribution of the faults (grouped by error type) made by each team. Similar to the data on errors, this data shows that all six teams also made faults that are classified into each of the three error types. We again ran one-sample t-tests to see if the frequency of these faults is significantly greater than 0. Again, in all three cases the number of faults was significantly greater than 0: *Slips* (p=.021), *Lapses* (p=.006), and *Mistakes* (p=.002).

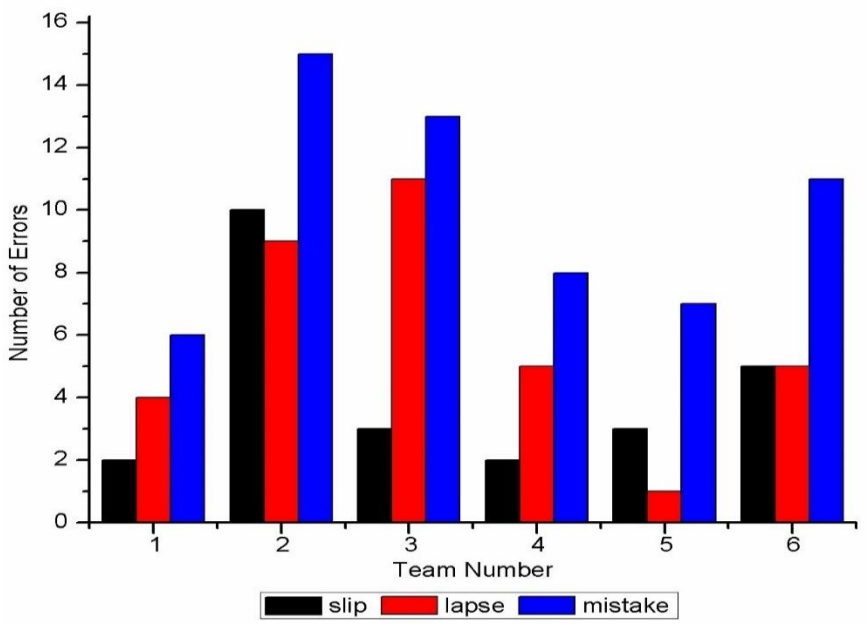


Figure 3.3. Team Errors by Error Type

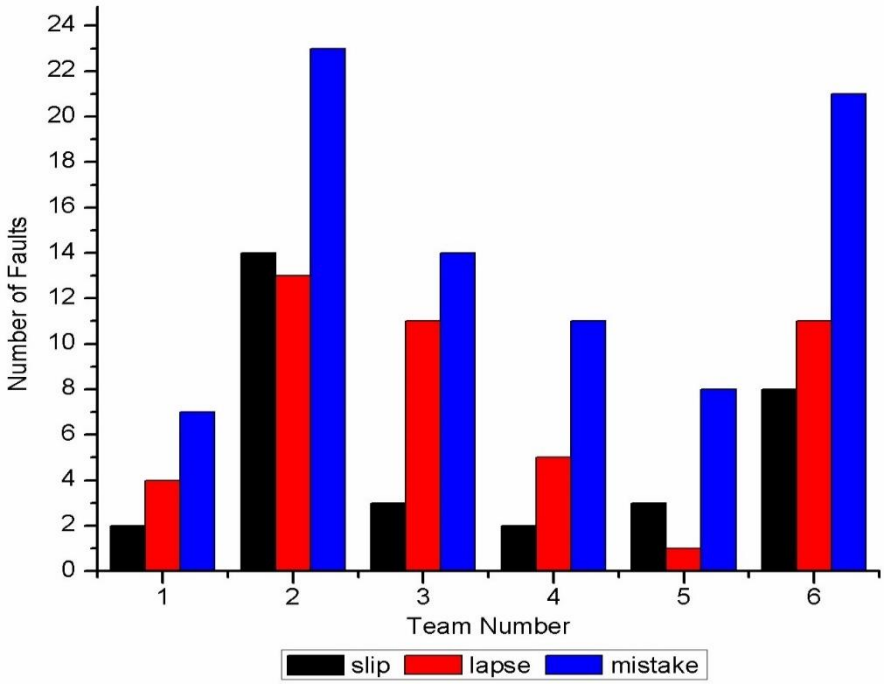


Figure 3.4. Team Faults by Error Type

Second, Table 3.2 shows the distribution of the errors among the 15 low-level error classes for each team. According to the table, eleven of the fifteen error classes were represented in at least one SRS document. Similar to the analysis of the error types, we conducted a series of

one-sample t-tests to determine if the number of faults in each class was significantly greater than 0 across all teams. The shaded cells in the last column of Table 3.2 indicate which error classes had an error frequency that was significantly higher than 0.

Table 3.3 shows the distribution of the faults (grouped by error class) for each team. Similar to the errors, this data shows that developers also make faults that are classified into most, but not all, of the error classes. Not surprisingly, the same error classes that were missing errors are also missing faults. The shaded cells in Table 3.3 indicate the error classes for which the frequency of errors was significantly greater than 0.

Table 3.2. Team Errors by Error Class

Error Types	T1	T2	T3	T4	T5	T6	% of Total Errors
CE	0	2	3	1	3	0	**8%**
LC	2	8	0	1	0	5	**13%**
LI	0	4	3	0	0	3	**8%**
AO	4	5	8	5	1	2	**21%**
AE	0	0	1	0	1	0	2%
EE	2	0	0	0	1	0	3%
IM	2	6	0	7	3	0	**15%**
WA	2	3	0	0	0	6	9%
LU	0	0	0	0	0	0	-
MB	0	0	10	1	1	0	10%
CD	0	0	0	0	0	0	-
LA	0	0	0	0	0	0	-
PS	0	0	0	0	0	0	-
IR	0	0	1	0	1	5	6%
SE	0	6	1	0	0	0	6%

Note: bold cells indicate result was significantly greater than 0

Table 3.3. Team Faults by Error Class

Error Types	T1	T2	T3	T4	T5	T6	% of Total Errors
CE	0	2	3	1	3	0	6%
LC	2	12	0	1	0	8	14%
LI	0	8	3	0	0	8	12%
AO	4	5	8	5	1	3	16%
AE	0	0	1	0	1	0	1%
EE	2	0	0	0	1	0	2%
IM	3	7	0	10	3	0	14%
WA	2	9	0	0	0	9	12%
LU	0	0	0	0	0	0	-
MB	0	0	10	1	1	0	7%
CD	0	0	0	0	0	0	-
LA	0	0	0	0	0	0	-
PS	0	0	0	0	0	0	-
IR	0	0	1	0	2	12	9%
SE	0	7	2	0	0	0	6%

Note: shaded cells indicate result was significantly greater than 0

H2: Developers will make more Mistake errors and faults.

Table 3.4 shows the percentage of errors and faults from each error class across all teams. The results of the Chi-square test (against a uniform distribution) shows that the distribution was significantly different from uniform. Examining the distribution shows that Mistake errors were responsible for half of the errors and faults.

Table 3.4. Comparison of Error Types

Variable	Slip	Lapse	**Mistakes**	P-value
Total Errors	21%	29%	**50%**	<0.001
Total Faults	20%	28%	**52%**	<0.001

3.4.2 RQ2: Do the errors identified by the HET lead to additional faults being found during reinspection?

This question focuses only on the value of the errors during the reinspection process. So, for this question we analyzed on the data reported on the *Reinspection Forms* to test this hypothesis.

H3: Development teams will find additional faults when reinspecting their SRS document guided by knowledge of existing errors (i.e., the observed frequency will be greater than 0).

During the **Reinspection Step** (Step 4 in Figure 3.2), members of each team inspected their own SRS guided by the errors identified in the **Error Abstraction Step** (Step 3 in Figure 3.2). Because the reinspection was guided by the already-identified errors, the teams were searching only for new faults (not new errors). Figure 3.5 shows the number of faults found during the reinspection by error type. The results show, similar to RQ1, faults related to the *Mistake* error type were the most common in the reinspection. The results of the one-sample t-test show that the frequency of faults found in the *Slip* and *Mistake* types were significantly greater than 0 ($p = .038$ and .007 respectively).

Figure 3.6 shows the number of faults found during the reinspection by error class. The one-sample t-tests did not show that the frequencies of any individual error class was significantly greater than 0.

3.4.3. RQ3: Do developers think the HET is helpful?

Using the data from the post-study survey, we tested the following hypothesis.

H4: Developers believe that the HET is useful for abstracting and classifying requirement errors.

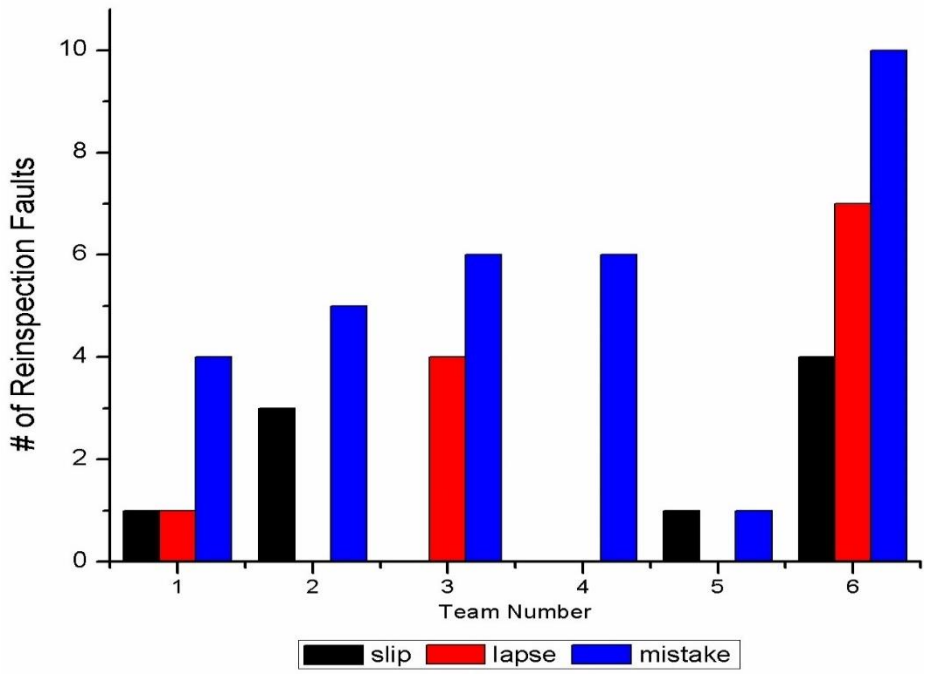


Figure 3.5. Faults found during Reinspection Grouped by Error Type

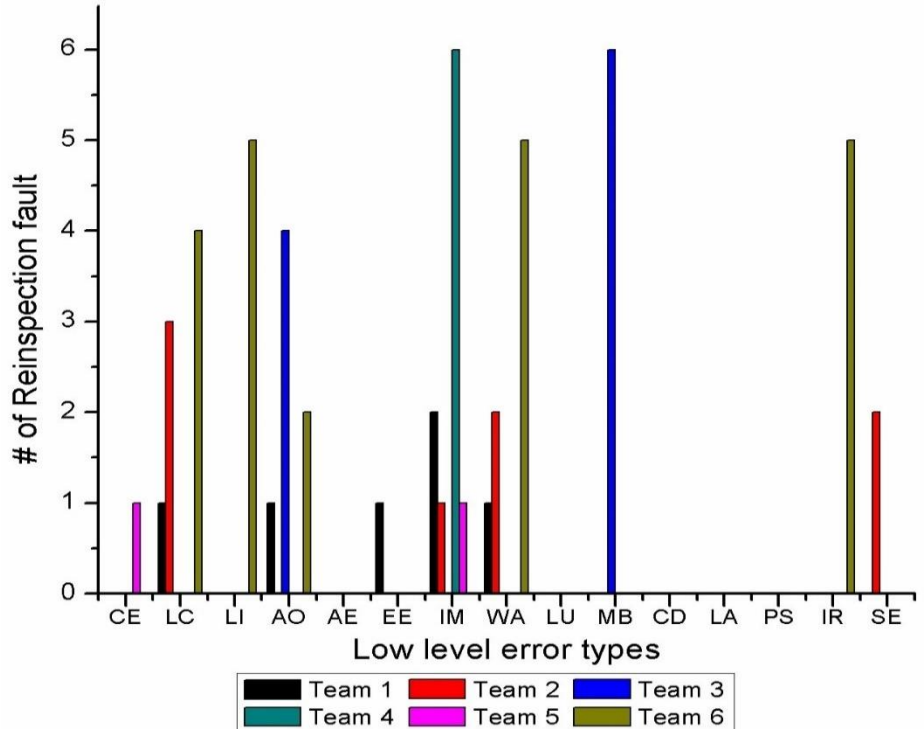


Figure 3.6. Faults found during Reinspection Grouped by Error Class

Table 3.5. Post-Study Survey

Construct (abbreviation)	Item	Questions
Usefulness	Usefulness 1	The HET is helpful for identifying faults
	Usefulness 2	The HET is complete
	Usefulness 3	The HET will be useful on future project
	Usefulness 4	The HET is helpful in improving the requirement documents
	Usefulness 5	The HET is helpful to detect overlooked faults
	Usefulness 6	The effort spent on using HET is valuable
Intuitiveness	Intuitiveness 1	The HET is intuitive
	Intuitiveness 2	I am confident that the errors identified represent real problem
Confidential (con)	Confidential	I am confident in the error abstraction process
Understandable (under)	Understandable	The HET is easy to understand
Classification (cla)	Classify	The HET is easy to use to classify errors
Abstraction (abs)	Abstract	The HET is easy to use to abstract errors
Hepful (help)	Helpful	The HET is helpful for understanding the faults
Orhogonality(ort)	Orthogonality	The HET is orthogonal
Clear training (clear)	Clear training	The training I received on HET is clear

Based on previous studies by Walia and Carver to obtain subjective feedback on the usefulness of the RET [9], we developed a post-study survey to evaluate nine specific characteristics of the HET: usefulness, intuitiveness, confidence, understandable, classification, abstraction, helpful, orthogonality and clear-training materials. Because we wanted to consider usefulness and intuitiveness a little more broadly than in the previous work, we added some dimensions to those two attributes. The participants were asked to use a 5-point scale ranging from 1 - strongly disagree to 5 - strongly agree to indicate their level of agreement about a series of statements related to these characteristics. Table 3.5 lists the statements included on the survey. As the survey was not mandatory, only 12 students responded.

3.4.3.1 Reliability and Validity

To make sure this survey is reasonable, we need to analyze its reliability and validity to reduce the uncertainty. The reliability of survey points to the extent of the consistent result of the same characteristic. We used the internal consistent to evaluate the reliability of our survey. Aiming to measure the internal consistency, we selected the average inter-item correlation and Cronbach's alpha to calculate the reliability of multiple-item measurements. As in our survey, we only have two characteristics (usefulness and intuitiveness) that used the multiple-items, so we just calculated the two construct reliability. Table 3.6 shows this result.

Table 3.6. Coefficient alpha of the scales

Construct	Number of items	Cronbach's Alpha	Average inter-item correlation
Usefulness	6	0.929	0.715
Intuitiveness	2	0.746	0.650

Based on the statistical theory, the value of α indicates the extent of reliability, with α < 0.7 means 'questionable' consistency, α >=0.7 means 'acceptable', α >= 0.8 means 'good', and α >= 0.9 means 'excellent'. From the result of Table 3.6 we can see that the survey is acceptable.

The construct validity of the survey evaluates how well the survey constructs measure what they are intended to measure. We used a principle components analysis with VARIMAX rotation to measure the convergent validity (whether different factors for the same underlying construct load onto the same factor) and divergent validity (whether the factors for similar but distinct concepts load onto different factors). In the results shown in Table 3.7, we highlight the factor for which each item has the highest loading factor.

Using this data, we can analyze the convergent and divergent validity. First, all items from the same theoretical construct loaded onto the same factor (F1- usefulness and F2 – intuitiveness). Second, statistical theory says that the eigenvalue for multi-item factors should be

at least .8, which our results satisfy for F1 and F2. Third, the smallest loading factor across all items is .708, which is in excellent range (>0.45 as fair, >0.55 as good, >0.63 as very good and >0.71 as excellent). Therefore, we can conclude that this scale has high construct validity, with regards to both convergent and divergent validity.

Table 3.7. Principle components analysis with VARIMAX rotation

Item	F1	F 2	F3	F4	F5	F6	F7	F8	F9
Usefulness 1	**.931**	-.096	.018	.205	.103	-.132	-.010	.157	-.029
Usefulness 2	**.916**	.055	-.055	-.112	-.13	.256	.017	.227	-.020
Usefulness 3	**.774**	.375	.154	-.059	.117	.066	.330	-.182	.261
Usefulness 4	**.761**	.372	.112	-.255	.44	.158	.334	.022	.066
Usefulness 5	**.760**	.157	.070	.127	.275	-.070	.532	-.043	.069
Usefulness 6	**.708**	.105	.305	.515	.050	-.111	.072	-.193	.241
Intuitiveness 1	-.041	**.892**	.004	-.162	.040	.207	.026	.049	.322
Intuitiveness 2	.428	**.822**	.307	-.008	.022	-.040	-.086	.074	-.148
Confident	.048	.181	**.920**	.020	-.02	.081	.177	.183	.208
Classification	.003	-.195	.011	**.906**	.113	.275	.094	.139	.152
Understandable	.137	.033	-.045	.099	**.910**	.270	.016	.250	.047
Abstraction	.057	.163	.097	.246	.291	**.897**	.051	.076	.053
Helpful	.398	-.154	.410	.177	-.05	.109	**.728**	.178	.191
Clear-training	.146	.120	.289	.137	.420	.100	.100	**.812**	.042
Orthogonality	.159	.239	.408	.319	.073	.069	.173	.050	**.773**
Total eigenvalue	6.228	2.25	1.97	1.55	.954	.710	.597	.325	.276
% of variance	41.52	15.0	13.1	10.3	6.35	4.73	3.98	2.17	1.83
Cumulative %	41.52	56.5	69.7	80.0	86.4	91.1	95.1	97.3	99.1

3.4.3.2 Frequency distribution of the nine characteristics

Figures 3.7, 3.8, and 3.9 show the distribution of the responses for each characteristic (Figure 3.7 has the characteristics with only one question each, and Figures 3.8 and 3.9 have the characteristics with multiple questions).

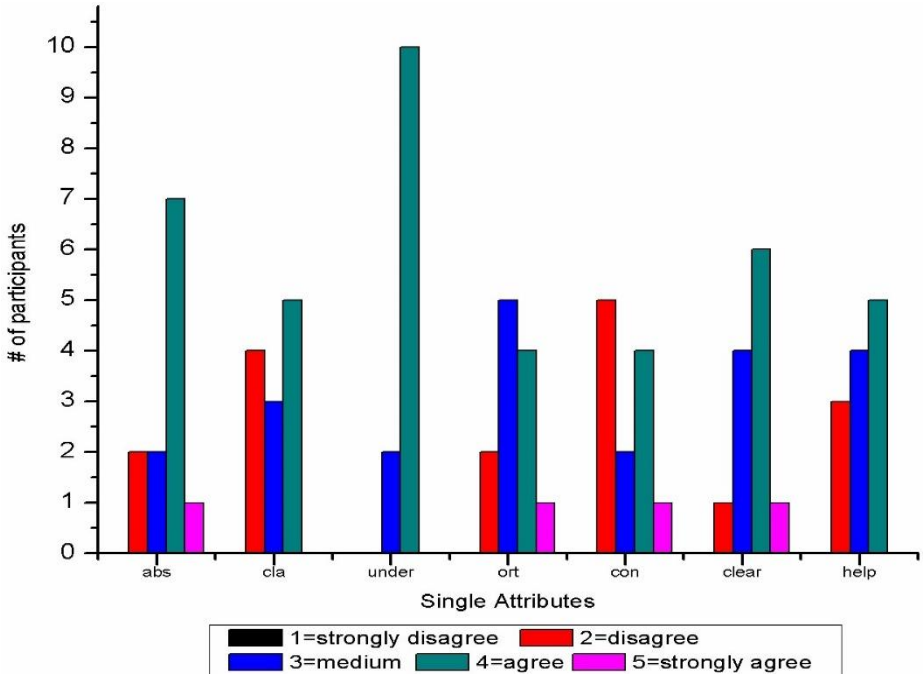


Figure 3.7. Responses for Single Attribute Characteristics

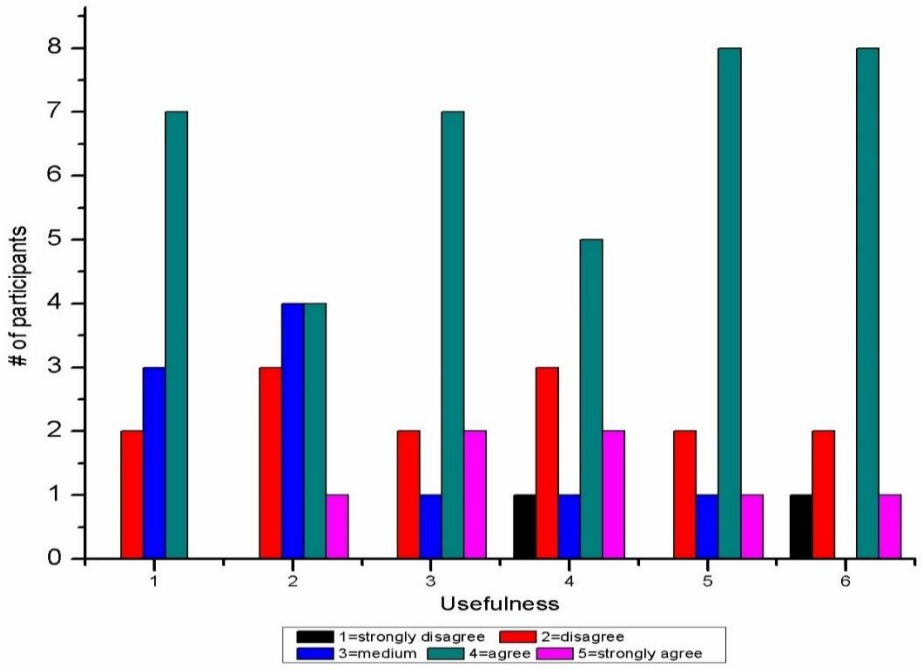


Figure 3.8. Responses for Usefulness Characteristic

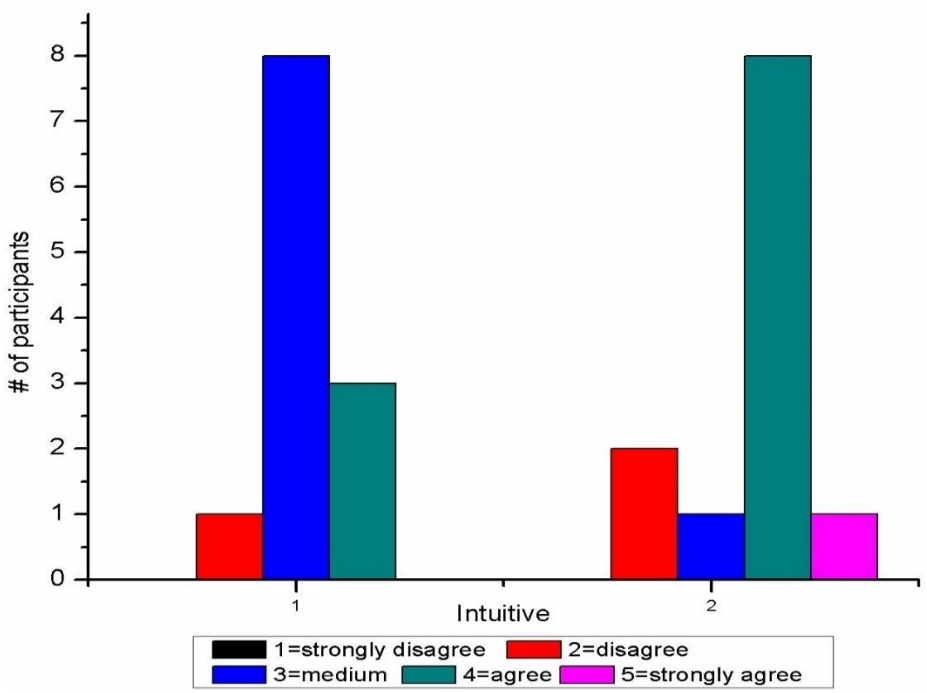


Figure 3.9. Responses for Intuitiveness Characteristic

The results show that, in general, the participants viewed the HET positively for all attributes. There are no cases for which an attribute received more low ratings than high ratings. To determine whether these attributes were rated significantly higher than the mid-point of the scale (3), we conducted a one-sample t-test for each characteristic. The results in Table 3.8 show that respondents viewed the HET significantly positive on most, but not all, attributes.

3.5. Summary and discussion of the results

The major focus of this study was to investigate the feasibility of using the HET to guide the inspection of SRS documents. If we can establish the feasibility of the HET, then future studies can provide more detailed evaluations of various aspects of the HET to ensure its usefulness in real development environments. To provide insight into that goal, this section provides answers to the three research questions posed in Section 3.1 with a detailed discussion and interpretation of the results in Section 3.3.

Table 3.8. Mean value of the ratings of the HET on different characteristics

Error types	p-value	df	t-statistic
abs	0.023	11	2.244
clas	0.377	11	0.321
under	0.00	11	7.416
ort	0.110	11	1.301
con	0.398	11	0.266
clear	0.014	11	2.548
help	0.252	11	0.692
usefulness 1	0.013	11	2.602
usefulness 2	0.047	11	1.820
usefulness 3	0.011	11	2.691
usefulness 4	0.197	11	0.886
usefulness 5	0.013	11	2.602
usefulness 6	0.012	11	2.691
intuitive 1	0.083	11	1.483
Intuitive 2	0.076	11	1.483

shaded cells indicate result was significantly greater than 3

3.5.1 RQ1: Does the HET provide a good method of describing and classifying the errors and faults made while developing the SRS?

This section examines the results in terms of the error types and the error classes. Then it describes some additional analysis to provide more insight into the results.

3.5.1.1 High-level Error Types

The results from Section 3.3.1 showed that the frequency of errors and faults committed is significantly higher than 0 for each of the error types. This result suggests that all three error

types in the HET are important because developers made errors and faults in each one of them. The results also showed that the most common type of error and fault was *Mistake*. There are three potential explanations for the higher number of *Mistakes*.

First, in the HET, there are more error classes in the *Mistake* error type than in the other two error types. Second, *Mistake* usually occurs because of a lack of knowledge, while *Slips* and *Lapses* tend to occur in familiar situations when attention is diverted. Finally, slips and lapses are often recognized and corrected quickly, and so may not make their way into the SRS documents, while mistakes tend to be more difficult to recognize and therefore persist longer. In this study, the participants were university students who lack experience and knowledge about software engineering in general and software requirements, specifically. Therefore, they may be more prone to *Mistake.*

3.5.1.2 Low-level Error Types

The results from Section 3.3.1 also showed that developers made errors and faults in most of the fifteen low-level error types. The most common error types are *lack of consistency in the requirements specification, loss of information from stakeholders, accidentally overlooking requirements, information management errors, and wrong assumptions*. Conversely, four error types: *Low understanding of one another's roles, Not having a clear distinction between client and users, Lack of awareness of sources of requirements, and Problem-solution errors* were not made by the developers in this study (at least there were no errors of these types identified during the inspections). From this one study, we are not yet ready to remove these error types.

It is possible that these were not relevant to the SRS documents developed in this study. It is also possible that developers made errors of these types, but just did not detect them during

the inspection process. We still need to conduct additional studies before making any final conclusions about the contents of the HET.

Further analysis of these results reveals that different teams made different errors and resulting faults. No single error class was responsible for the majority of the errors or faults. For example, Teams 2 and 4 made a large number of *Information Management* errors and faults, while the majority of Team 3's errors and faults resulted from the *Mistaken* belief that it is impossible to specify non-functional requirements in a verifiable form and Team 6 made the majority of its errors and faults because of an *Inadequate Requirements Process*.

Another interesting observation is that some error classes are more common across all teams than others. For example, all teams made an error and related fault because of *Accidentally Overlooking Requirements*. Four of the six teams made errors and faults related to *Clerical Errors, Lack of Consistency in the Requirements Specification, and Information Management*. Other error types were not present at all or were only present on two or three teams. These observations suggest that the SRS development process varies across teams leading different teams to make different errors and faults. These team differences could arise through many different reasons. Individual developers may have specific strengths and weaknesses in software development, team dynamics may play a role, and the characteristics of the system under design may place distinct strains on the development effort. Because our teams all worked on different projects with different members, we cannot analyze the contributing roles of such factors.

3.5.1.3 Additional Analyses

Our analysis of the results showed that three teams (T1, T3, and T6) did not classify all abstracted errors into the HET. To complete the dataset and ensure that no information was omitted, I classified those errors into their appropriate error classes and types. We report these

results here as a separate analysis rather than as part of the main results since we did the classification rather than the participants. This step resulted in 18 additional errors (9 in Team 1, 2 in Team 3, and 7 in Team 6) and 12 corresponding faults (10 in Team 1, 2 in Team 3, and 9 in Team 6). This subjective classification did not change the overall distribution of errors or faults.

3.5.2 RQ2: Do the errors identified by the HET lead to additional faults being identified during reinspection?

One of the goals of this feasibility study was to understand whether knowledge of errors would aid developers in finding additional faults related to those errors. The main idea is that once a team becomes aware that they have committed an error (that is some human mental failing) in the SRS development, it is likely that additional faults related to that error are also present in the SRS. The amount of time that development teams can devote to inspections may be limited. The results from Section 3.3.2 showed that all teams found additional faults during the reinspection. If we can show that knowledge of errors can lead to identification of additional related faults, then teams might be able to better focus their effort during a reinspection task.

One of the limitations of the feasibility study is the lack of a control group. The goal of this study was not to compare HET to another approach. As a result, we cannot draw a conclusion about whether development teams would have found more or less faults during a reinspection without guidance from the error information. The goal of this study was to show that use of error information could lead to detection of additional faults. The next step in our work is to conduct control group studies to determine whether the insights provided by the HET provide more assistance to a developer than other approaches (e.g., a simple reinspection using the same fault checklist as the first inspection).

3.5.3 RQ3: Do developers think the HET is helpful?

In order for an approach like the HET to be adopted, it must show objective value (RQ1 and RQ2) and subjective value. That is, if developers do not think the approach is helpful or do not like the approach they are less likely to use it, regardless of whether the objective data shows it to be effective. The results from Section 3.3.3 showed that overall the participants had a favorable view of the HET. Most participants either agreed or strongly agreed with each characteristic. The participants indicated that the HET was helpful for abstracting and classifying faults into errors and that it was helpful for identifying additional faults during the reinspection. We did not gather any qualitative feedback through interviews or additional survey questions. In our future studies, we will gather more detailed feedback to allow us to improve the HET.

3.6. Validation of Results with Industrial Data

The classroom study provided initial evidence that the HET is useful for requirements inspections. It is important to validate whether those classroom findings will be valid in an industrial setting. The goal of this industrial validation is to understand whether the HET describes requirements errors that occur in practice and to get feedback on the HET from the industrial professionals. To achieve this goal, I propose the following research question:

RQ4: Does the HET describe real problems faced by practitioners?

3.6.1 Data Source

To investigate RQ4, I conducted interviews with professional developers at CAPS (the Center for Advanced Public Safety, a software development organization contained within the College of Engineering at UA) and analyzed data from the NaPiRE (*Naming the Pain in Requirements Engineering*) survey, a large-scale industrial survey conducted by collaborators to

gather data regarding the sources of requirements problems. In the following sections, I describe each of these data sources in detail.

3.6.1.1 CAPS data

The first data source is the results of interviews I conducted with developers from CAPS at the University of Alabama. Those developers were all involved in the requirements engineering process. As background, the CAPS organization builds different types of systems for different customers. The main systems that they developed are for traffic safety, homeland security, motor vehicles, analytics, health and human services, weather and disaster responses, emergency medical services and law enforcement. They also work with many Alabama state agencies as well as agencies in other states. In developing these system, CAPS has a core staff of business analysts, software engineers, and software developers who work closely with clients to ensure that required features are implemented.

I chose CAPS developers as a data source due to their rich experience about software requirements development. Their experiences provide useful industry data to support my analysis. I interviewed seven developers who had experience in requirements development or requirements evaluation. During the interview, I first briefly introduced requirements errors and their importance in the software development process. Then, I asked participants to describe any commonly occurring requirements errors they experienced on their projects along with any prevention mechanisms they employed or planned to employ. Next, I introduced the HET with a detailed explanation of each high-level error types and low-level error classes. Finally, I asked the interviewee whether any of the errors he/she described could be classified into the HET.

3.6.1.2 NaPiRE dataset

The second data source is the data from the NaPiRE survey. The main idea of NaPiRE is to conduct a broad survey investigating the status quo of requirements engineering in practice along with the problems requirements practitioners face. This survey is conducted bi-annually with practitioners across the world.

The data used here is drawn from respondents in 10 countries representing 226 companies (one participant per company). The survey presented 21 common requirements problems and asked the participants to choose the top 5 based on their experiences. For each of the 5 selected problems, the respondent listed the possible causes of the problem and the mitigation approaches they have taken or plan to take to address that problem [12].

As the topic of this survey and the data gathered are highly relevant to our work, i.e., providing information about the types of problems faced by requirements engineers and the sources of those problems, we have begun collaborating with the NaPiRE team. Here, I use the NaPiRE data to do a deep analysis to determine how well the HET describes the sources of requirements problems and to validate whether the HET is complete with regards to these real problems.

3.6.2 Procedure

In this section, I describe the process followed to analyze the data. To ensure a valid analysis of the data, this process included three reviewers.

1. Me – with knowledge of software engineering and human error taxonomies;
2. Dr. Jeffrey Carver – a software engineering expert with experience in empirical data analysis; and
3. Dr. Gary Bradshaw – a cognitive psychologist with expertise in human error analysis.

This group of reviewers provides expertise from different perspectives to help ensure a broad coverage of the results and a valid classification of the data. We analyzed the data from each data source separately following the same six steps, described as follows:

Step 1 - Individually review data: The CAPS interviewees described a total of 45 requirements errors commonly faced in their development experience. The NaPiRE survey results provided a total of 92 requirements errors for the given requirements problems. For each dataset, each reviewer individually each requirement error or cause to determine whether it described a human error (slips/lapses/mistakes) or another type of problem (not a human error). During the analysis process, we also used information provided about mitigation strategies to help us better understand the participants'' descriptions. The output of this step is *six individual data forms*, three for the CAPS data and three for the NaPiRE data.

Step 2 – Comparison of the individual results: Using the *individual data forms* for each dataset (CAPS and NaPiRE), I combined the responses into a consolidated *data analysis form* (one for each dataset). Then, I marked any differences among the results from the three reviewers.

Step 3 – Consolidation of the individual high-level results: In this step, the three reviewers met together to discuss the differences in their initial categorization of the errors. Based on these discussions, we resolved any discrepancies and recorded the final results into a spreadsheet for each dataset.

Step 4 – Categorization the low-level error types: After removing items that were not human errors, I attempted to map the remaining items into the low-level error classes of the HET. To perform this mapping, I used both the description of the error and the mitigation strategy provided by the respondents. The mitigation strategies were helpful in understanding

more clearly what the respondent meant for each described error. In cases where respondents provided different mitigation strategies, I mapped each one to the appropriate human error class. For errors that I could not map to the HET, I marked them as new human errors. For errors that did not include enough information to fully understand them, I marked them as 'Cannot be categorized'. The output of this step was two forms, one for each dataset.

Step 5 – Check the correctness of Step 4: The other two reviewers checked results of my analysis in Step 4 to determine whether they agreed with my categorization of the low-level error types for identified human error. Then marked any disagreements found.

Step 6 – Consolidate the low-level results: Finally, the three reviewers met together to discuss the differences identified during Step 5 and to finalize the low-level categorization of the human errors.

3.6.3. Results

This section describes the detailed analysis results for the research question: *RQ4: Does the HET describe real problems faced by practitioners?*

The results of the analysis process described above led to the elimination of 16 of the responses in the CAPS dataset because they were not truly human errors. Table 3.9 shows how we classified the remaining 34 human errors into the HET. We were able to map 28 of the 34 errors into the HET, leaving 6 mistake errors that required addition of a new error type.

Table 3.9. Distribution of Error Types of CAPS Data

High-level Error Types	Low-level Error Classes	Count
Slips (1)	Clerical errors	1
	Term substitution	0
Lapses (5)	Loss of information from stakeholders errors	5
	Accidentally overlooking requirement errors	0
	Multiple terms for the same concept errors	0
Mistake (28)	Application errors: knowledge-based plan is incomplete	2

	Environment errors: knowledge-based plan is incomplete	0
	Solution Choice errors	0
	Syntax errors	0
	Information Management errors: knowledge-based plan is incomplete	0
	Wrong Assumption errors: knowledge-based plan is wrong	1
	Mistaken belief that it is impossible to specify non-functional requirements: knowledge-based plan is incomplete	0
	Not having a clear distinction between client and users	0
	Lack of awareness of requirement sources: knowledge-based plan is incomplete	0
	Inappropriate communication based on incomplete/faulty understanding of roles: knowledge-based plan is wrong	11
	Inadequate Requirements Process	8
	New error type - Management error	6

As Table 3.9 shows, *Mistake* errors are the most commonly identified high-level error type. Among these *Mistake* errors, only four of the eleven low-level types are represented. Similarly, for the *Slip* and *Lapse* errors, the participants identified errors in only one of the low-level error types. Based on the provided data, we were able to add a new Mistake error type – a *management error*. In our sample, we identified different manifestations of the *management error*, not clearly defining team roles and responsibility, not properly allocating resources, not providing proper training for RE team members, and not having a good plan for changing requirements.

Table 3.10. Distribution of Error Types of NaPiRE Data

High-level Error Types	Low-level Error Classes	Count
Slips (1)	Clerical errors	0
	Term substitution	0
	Cannot classify	1
Lapses (0)	Loss of information from stakeholders errors	0
	Accidentally overlooking requirement errors	0
	Multiple terms for the same concept errors	0
Mistake (64)	Application errors: knowledge-based plan is incomplete	1

	Environment errors: knowledge-based plan is incomplete	1
	Solution Choice errors	0
	Syntax errors	0
	Information Management errors: knowledge-based plan is incomplete	6
	Wrong Assumption errors: knowledge-based plan is wrong	1
	Mistaken belief that it is impossible to specify non-functional requirements: knowledge-based plan is incomplete	2
	Not having a clear distinction between client and users	0
	Lack of awareness of requirement sources: knowledge-based plan is incomplete	0
	Inappropriate communication based on incomplete/faulty understanding of roles: knowledge-based plan is wrong	14
	Inadequate Requirements Process	19
	New error type - Management error	20

Secondly, we did the same analysis for the NaPiRE survey data. The results of the analysis process described above led to the elimination of 27 of the responses in the NaPiRE dataset because they were not truly human errors. Table 3.10 shows how we classified the remaining 65 human errors into the HET. We were able to map 45 of the 65 errors into the HET, leaving 20 mistake errors that required the addition of a new error type.

From Table 3.10, we can find that most of the human errors that these participants provided are still mistake errors. Among these mistake errors, there are seven low-level error types that appear at least once.

3.6.3 Discussion of results

Based on these datasets, we observe that respondents report slips and lapses less frequently than mistakes during the requirements engineering process. One reason for this difference in frequency is that even though slips and lapses likely occur as frequently as mistakes, people tend to catch slips and lapses as they occur and quickly, even subconsciously, fix them, rather than letting them become a problem. Therefore, when people describe the types

of errors that occur, they are more likely to recall the mistakes because they had to take explicit action to address them.

Next, the participants experience and the data collection process are different between the interview with CAPS and the NaPiRE survey, thus, to analyze whether the distribution of errors between the CAPS and NaPiRE datasets are different, we performed a Chi-square analysis. The test results showed that chi-value is 20.46, p-value is 0.015, df is 9, which means that the distribution of error types between these two studies is different. We can pose the potential reasons for the different distributions is that the CAPS dataset contains data from only 7 participants who were asked to describe any errors they could recall. Conversely, for the NaPiRE data, there were a large number of respondents who were asked to provide the reason why a selected set of problems occurred.

In order to perform a detailed analysis of the difference between these two datasets and to understand why some Mistake error types are more common than others, we divided the Mistake errors into two groups: internal errors and interface errors. Internal errors originate from within a person and include concepts like their personal skills or knowledge. Conversely, interface errors focus on the interfaces or interactions between the requirements engineer and other stakeholders, including those managing the requirements engineering process. Using these definitions, we classify the following error types as interface errors: information management errors, inappropriate communication based on incomplete/faulty understanding of roles, and the new error type -management errors. The remaining error types are internal errors.

In the CAPS dataset, 17 of 28 Mistake errors are interface errors. Similarly, in the NaPiRE dataset, 40 of the 64 errors are interface errors. In both cases, interface errors represent the majority of the errors. Therefore, we can conclude that because interface errors seem to be

more prevalent, developers need to pay special attention to communication problems to reduce the presence of these errors.

The result also shows that four mistake errors did not show up across both datasets: *Solution Choice errors*, *Syntax errors*, *Not having a clear distinction between client and users*, *Lack of awareness of requirement sources: knowledge-based plan is incomplete*. These four error classes have a common characteristic: they are all internal Mistake errors and can be avoided by increasing the requirements engineers' knowledge and experience.

3.7 Comparison of classroom study and industrial results

Based on the results from the classroom study and the industrial studies, we can make the following comparisons:

- *Distribution of error types* -- Both studies showed that **Mistake** errors are the most common. Comparing the results of the two studies shows that three low-level error types are missing from both samples: *Not having a clear distinction between client and users, Lack of awareness of sources of requirements, and Problem-solution errors.* To compare the distribution of classroom study and industrial results, we performed a chi-square test for the data in Table 3.2 (sum of errors among all the six teams for each error type) with the combined error distribution count of CAPS and Interview data in Table 3.9 and Table 3.10. The results of the test ($\chi^2_{12} = 43.30$, $p < .01$) showed that the two samples had significantly different distributions. One potential reason for the difference is that the experience level differs between the students and the professionals. It should be expected that professionals encounter different errors on real projects than students do on classroom projects. Another reason is that for the industry studies, we just asked participants to provide the requirements problems based

on their past real system development experience, just based on their memory, the results still not rogue.

- *Representation of HET* -- In the classroom study, the participants were able to abstract and categorize all faults into the HET. Conversely, in the industry study, we could not map some requirements errors into HET. Therefore, we added a new error type – management error. This difference likely results from the fact that in industry the larger development teams require more management process to make the effective. This characteristic may not have been evident in the smaller teams used in the classroom setting.

3.8. Threats to Validity

We faced the following threats to validity in one or both studies.

Internal Validity There are two primary internal validity threats: mortality and lack of a control group. Regarding mortality, for the feasibility study, only six of the eight teams submitted valid data. Even though it is possible that the results would change if we had data from all eight teams, we have no evidence to suggest that the two missing teams would have behaved any differently. Furthermore, only 12 participants responded to the post-study survey. It is possible that only those participants who viewed the HET as helpful took time to respond to the survey. But again, we have no evidence that those who responded are different from those that did not. The same validate existed in the survey data, we only obtained 7 participants. For survey study, it does not have the mortality validity.

For the feasibility study, the lack of a control group threat manifested itself in two ways. First, because we did not have developers use other approaches for classifying errors (like the RET) we cannot conclusively determine that the HET is the most effective approach. Second,

because we did not have a control group perform a reinspection using some other approach, like a second fault-based inspection, we cannot be sure how many of the additional faults found during the reinspection were truly due to the use of the HET. For the survey study, we also did not analyze the RET to see whether HET can better describe the real problem or not. We will eliminate this threat by adding a control group in the future study.

External Validity In the feasibility study, the participants were building a realistic system to solve a problem they encountered in their own lives. However, the participants were still undergraduate students in a classroom environment, rather than professional developers in a real environment. We deal with this problem in the interview and survey study by collecting the professional developers as participants.

Construct Validity For the field study, because the SRS documents were developed as part of the study, we do not know the real number of faults and errors present. Therefore, our conclusions are based only on the errors and faults actually detected. In the interview and survey study, all these data are subjective and based on the participants' experience, our conclusion still have this kinds of validity. In the future, we will develop some experiment study with industry participants to eliminate this threat.

3.9. Conclusion and future work

The results of the studies described in this chapter show a positive impact of the HET both objectively and subjectively. Objectively, developers were able to abstract errors from faults, classify those errors into the HET, and use the classified errors to find additional faults during a reinspection. Subjectively, the participants rated the HET positively on a number of key characteristics.

Regarding the HET itself, both studies showed that *Mistakes* were the most common type of requirements error, specifically interface errors. This finding may be caused by two reasons: 1) in our HET, there are more Mistake errors than Slips and Lapses, thus more errors may map to Mistakes; 2) the requirements engineering process includes several stakeholders who must communicate and be managed, thus resulting in a large number of interface errors.

Additionally, all but three of the low-level error classes were present in at least of one of the two studies. These results indicate that overall the HET contains the right types of errors to describe the problems that occur during SRS development.

Furthermore, with the industry dataset, we found a new error type, management errors, that we could not map into the existing HET. Therefore, in the future, we need to update our taxonomy, remove some error types that do not appear in either study and add a new class (mistake error – management error) based upon the results of the industrial studies.

In the future, we will do another real industry study with professional developers, to ask them to use our HET to detect the requirements errors in their requirements documents, and then to see the distribution of error types, and to see whether these three low-level error types also do not have any contribution or not. Based on the results that we obtained from these three types of studies, we will update our HET error types.

References

[1] Boehm, B., Basili, V. R.: Software defect reduction top 10 list. Foundations of empirical software engineering: the legacy of Victor R. Basili, 2005: 426-428.

[2] Carver, J.: The impact of background and experience on software inspections. Empirical Software Engineering, 2004, 9(3): 259-262.

[3] Chillarege, R. I., Bhandari, S. Chaar, J. K., Halliday, M. J., Moebus, D. S., Ray, B. K., Wong. M. Y.: Orthogonal defect classification-a concept for in-process measurements. IEEE Transactions on Software Engineering, 1992, 18(11): 943-956.

[4] Freimut, B., Denger, C., Ketterer. M.: An industrial case study of implementing and validating defect classification for process improvement and quality management. 11th IEEE International Symposium in Software Metrics, 2005: 9-15.

[5] Hayes. J. H.: Building a requirement fault taxonomy: Experiences from a nasa verification and validation research project. 14th International Symposium on Software Reliability Engineering, 2003: 49-59.

[6] Kraemer, S., Carayon. P.: Human errors and violations in computer and information security: The viewpoint of network administrators and security specialists. Applied Ergonomics, 2007, 38(2): 143-154.

[7] Lanubile, F., Shull, F., Basili, V. R.: Experimenting with error abstraction in requirements documents. Proceedings. Fifth International in Software Metrics Symposium, 1998: 114-121.

[8] Lawrence, C., Kosuke, I.: Design error classification and knowledge management. Journal of Knowledge Management Practice, 2004, 10(9): 72-81.

[9] Leszak, M., Perry, D. E., Stoll. D.: A case study in root cause defect analysis. In Proceedings of the 22nd International Conference on Software Engineering, 2000: 428-437.

[10] Lopes, M. E. R. F., Forster, C. H. Q.: Application of human error theories for the process improvement of requirements engineering. Information Sciences, 2013, 250: 142-161.

[11] Masuck. C.: Incorporating a fault categorization and analysis process in the software build cycle. Journal of Computing Sciences in Colleges, 2005, 20(5): 239-248.

[12] Mendez, F. D. et al.: Naming the pain in requirements engineering. Empirical Software Engineering, 2016: 1-41.

[13] Porter, A. A., Votta Jr, L. G., Basili, V. R.: Comparing detection methods for software requirements inspections: A replicated experiment. IEEE Transactions on Software Engineering, 1995, 21(6): 563-575.

[14] Reason. J.: Human error. Cambridge university press, 1990.

CHAPTER 4

EVALUATING USE OF HUMAN ERROR INFORMATION FOR ERROR PREVENTION

4.1. Introduction

The quality of software products largely depends on the quality of the underlying requirements. Prior research has shown the importance of producing correct requirements because requirements faults are more expensive to fix later [8], are among the most severe kinds of faults [6], and cause the majority of software failures [13]. Due to the large expense to find and fix faults after they occur, it is crucial to develop effective defect prevention methods.

The software development process, especially during the requirements phase, is a human-centric activity. As humans are fallible, the potential for error is high. As defined by IEEE Standard 24765 [1], an error is the failing of human cognition in the process of problem solving, planning, or execution. These cognitive failures can then lead to various types of requirements faults. Cognitive psychologists have long studied these cognitive failures and referred to these cognitive failures as the term human errors. By understanding how the human mental process can fail in various situations, human error research has been able to support error prevention in fields ranging from medicine to aviation.

Of all the software engineering phases, the requirements engineering phase may be the most human-centric. Therefore, the ability to understand and prevent human errors that occur during requirements engineering can be especially beneficial to software projects. To make this human error information tractable, human error researchers develop taxonomies to classify the specific

types of errors that occur in each domain. While the underlying theoretical basis is similar across domains, the specific types of errors differ. In our own previous work, we have developed two taxonomies of requirements errors, using different approaches:

The Requirement Error Taxonomy (RET) [25] and the Human Error Taxonomy (HET) [15]. Section 2.3 provides more details on these taxonomies.

Because human errors occur while eliciting and formalizing requirements, we anticipate that as requirements engineers better understand specific types of errors, the less likely they will be to make those errors, resulting in higher-quality requirements. Therefore, the goal of this research is to evaluate whether an understanding of Human Error reduces the likelihood of making errors and

the resulting faults during the requirements engineering process, and what kinds of prevention mitigation we can provided for the human errors.

The primary contribution of this paper are (1) evaluation of whether the knowledge of error information prevents developers from injecting related errors and faults into a requirements document, (2) comparison the performance of RET and HET in providing guidelines for developers in developing requirements document, (3) an analysis of contribution of the specific error types in RET and HET in preventing errors and faults, and (4) provide the detail prevention mitigation for all the human errors in the HET.

The remainder of this paper is organized as follows. Section 4.2 describes related work. Section 4.3 provides a description of the study conducted to evaluate the utility of error information in preventing errors and faults. Section 4.4 describes the analysis and results of this study. Section 4.5 describes the prevention mitigation for each low-level error types. Section 4.6

discusses the threats to validity of this study, followed by a brief conclusion of this paper and ideas for future studies in Section 4.7.

4.2. Background

This section provides background information on topics relevant to our study. Section 4.2.1 describes previous research about fault prevention. Section 4.2.2 briefly describes research related to human error. Section 4.2.3 briefly describes the two kinds of error taxonomies (RET and HET) that we have developed.

4.2.1 Fault prevention techniques

The fault prevention process uses information about the types of problems that are likely to occur (often using historical data, a sample of faults, or expert opinion) to prevent those problems from occurring in the future. Defect prevention methods utilize the fault injection rate in software development processes and provide specific strategies to prevent related faults [12]. Furthermore, training and mentoring can also result in dramatic reductions of fault rates (close to 50%) [4]. There are three types of fault prevention measurement recently applied.

- Quality improvement techniques - These techniques, that have seen widespread success [5, 10, 14, 19], focus developer's attention on different type of faults (e.g., missing or incorrect functionality) recorded in software artifacts. Because they only deal with the fault itself, these techniques cannot help inspectors understand underlying errors (i.e., source of faults) and prevent the occurrence of these errors.
- Prevention of faults via process improvement - These methods focus on determining the causes for commonly identified faults. Examples include:
  - fault causal analysis [4] - uses a sample of faults to determine their causes and prevent future faults. An empirical study found that most of the causes of software faults were due to people-related factors;

- software failure analysis [11] - improves the development and maintenance process by analyzing a representative sample of faults to understand the causes of particular classes of faults;
- fault prevention process [20] - determines the source of a fault and suggests preventive actions by classifying faults causes as oversight (e.g., developer overlooked something, or something was not considered thoroughly), education (developer did not understand some aspect), or transcription (developer knew what to do but simply made a mistake).
- root cause analysis [18] - uses a multi-dimensional fault trigger concept to help developers determine the root cause of a fault and help them identify improvement area.

- Error abstraction approach - Lanubile et al. proposed the process of analyzing a group of related faults to determine the errors that lead their occurrence [17]. Based on this approach, several fault prevention techniques have been proposed, including: the fault distribution method [21], fault based process improvement (DBPI) technique [16], and fault prevention-based process improvement (DPPI) method [23]. However, these methods lack formal error information to provide guidelines for developers to use. They can only prevent the errors that have been identified, but cannot provide the type of information that developers need to learn from those errors.

While all these methods use some representative faults/problem reports to analyze the root cause, they lack an underlying cognitive theory of how people make errors. Therefore, there was a need to develop more formal taxonomies, like the RET and HET, to address this shortcoming.

4.2.2 Human Error Research

Human error research focuses on understanding how the psychological processes go awry. For example, choosing an incorrect solution, forgetting to perform a task, or accidentally performing an incorrect task. The process of analyzing human error in a particular domain includes collecting information, finding common failure patterns, and interpreting those patterns in light of the limitations of human information processing facilities and known error patterns. This type of understanding can provide insight into how to prevent similar errors from happening in the future. This approach has been used successfully to improve aviation [26] and medicine [9].

4.2.3 Error Taxonomies

To make the error information described above useful, researchers classify the errors into taxonomies. A taxonomy provides a logical, hierarchical organization of the error information. Without such an organization, developers will find it difficult to successfully use the information about errors to have a practical impact on their work. Earlier work on using error information to improve software quality [7, 17, 18] provided developers with ways to use the sources of faults (i.e. errors) to improve software quality. While this work provided a significant step forward, the main weakness was the lack of a formal error taxonomy. To improve upon these methods, we employed two different approaches to create taxonomies of human errors that occur in the requirements phase. The remainder of this section explains each taxonomy in more detail.

Requirement error taxonomy (RET) To develop the first taxonomy, we performed a systematic literature review (SLR) to identify and classify requirements errors that were described in the software engineering and cognitive psychology literatures. We developed this taxonomy strictly from the software engineering perspective without input from a human error

expert. Therefore, we built it bottom-up, based primarily on the errors identified in the literature, without

using a formal human error theory as a driver. Figure 2.1 provides an overview of the taxonomy, which includes 14 detailed error classes grouped into three high-level error types. The three high-level error types are: People Errors (arise from fallibilities of the people involved in the development process), Process Errors (arise while selecting the appropriate processes for achieving the desired goals and relate mostly to the inadequacy of the requirements engineering process), and Documentation Errors (arise from mistakes in organizing and specifying the requirements) [25]. Initial evaluation of the RET demonstrated that finding more errors during a training exercise resulted in fewer errors during requirements creation [24], suggesting that error information can be helpful for prevention.

Human Error Taxonomy (HET) To more closely tie our work to the concept of human error, we collaborated with a human error expert to develop a new taxonomy. We again performed an SLR to identify specific requirements engineering human errors reported in the literature. But, this time, we classified those detailed errors into a predefined set of high-level error types drawn from human error research [22] (see Figure 2.3). Those three high-level error types are: Slips (someone carries out a planned task incorrectly or in the wrong sequence), Lapses (memory related failures; they occur when someone forgets a goal in the middle of a sequence of actions or omits a step in a routine sequence), and Mistakes (planning errors in that someone designed an incorrect plan to achieve the desired goal). The details of the HET are beyond this paper, but have been described elsewhere [2, 15].

We have conducted a feasibility study to validate the effectiveness of HET for detecting human errors and faults, but not for prevention [15]. The results of this study showed that the use

of the HET allowed developers to identify errors in requirements documents and to find additional faults during a reinspection guided by knowledge of the existing errors. Thus, in this study, we plan to evaluate the usefulness of HET in preventing human errors.

4.3. Experiment design of control group study

The goal of this study is to evaluate whether an understanding of Human Error helps prevent faults during requirements creation. The following subsections describe the research questions, the participants, and study procedures.

4.3.1 Research Hypotheses

To address the overall study goal, we investigate three specific hypotheses. These hypotheses move from human errors in general to more specific aspects of human error. Figure 4.1 provides an overview of the relationship among the hypotheses.

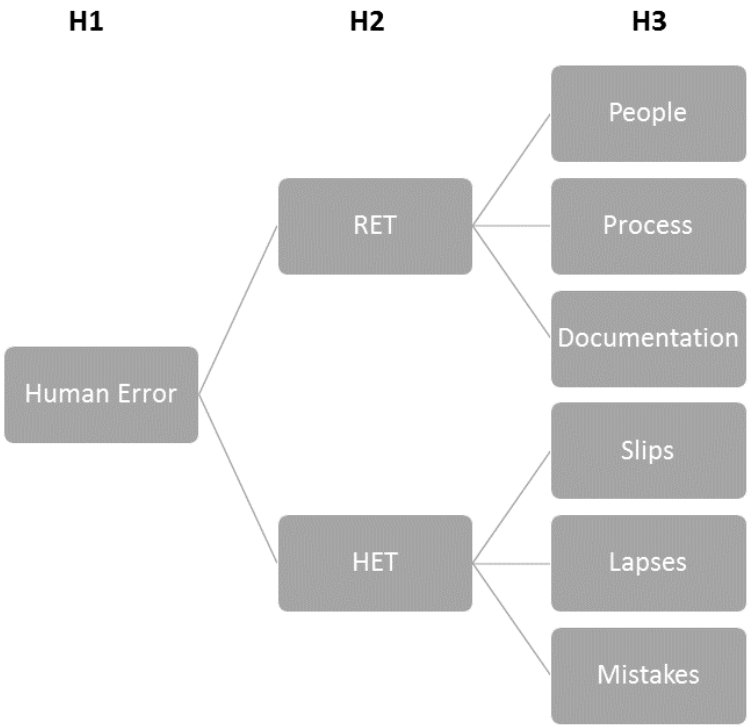


Figure 4.1. Research Questions

First, our previous work indicated that knowledge of requirements errors can help in fault prevention (see Section 2) [24]. In this study we employ two different methods for understanding requirements errors (the RET and the HET). Based on previous work that the information could be useful, our first hypothesis is:

H1 *- The better a developer understands human errors the less likely he/she will be to inject errors and faults into a requirements document*

Second, because we built the HET on a stronger cognitive theory of human errors (compared with the RET), it should provide more help with regard to preventing errors and faults. Therefore, our second hypothesis is:

H2 *- Knowledge of the HET will provide more benefit for error/fault prevention than knowledge of the RET*

Finally, the RET and the HET each have three high-level error types that represent different types of human errors. We anticipate that the better a developer understand each type, the fewer of that type of error he or she will make. Therefore, our third hypothesis is:

H3 *- The better a developer understands each error type, the less likely he/she will be to insert errors/faults related to that type into a requirements document.*

4.3.2 Variables

For each hypothesis, this section describes the Independent and Dependent variables. Section 4 defines how each variable is measured.

- H1
  - Independent Variables
    - Ability to classify errors
    - Ability to use error information to find new faults
  - Dependent Variable

- Likelihood of injecting new faults

- H2
  - Independent Variable
    - Knowledge of error taxonomy (HET or RET)
  - Dependent Variables
    - Number of faults injected into SRS
    - Number of errors injected into SRS

- H3
  - Independent Variable
    - Level of understanding of each error type in the HET or RET
  - Dependent Variable
    - Number of each error type injected into SRS

4.3.3 Participants

The study included 31 senior-level undergraduate computer science enrolled in the Spring 2016 capstone course at the University of Alabama. In this course, students worked in teams to iterate through the software lifecycle and build a software system. The course instructor, independent of the research team, divided the participants into ten 3- or 4-person teams. To address H2, we randomly assigned each team to either the RET group (control) or the HET group (experimental). Table 4.1 illustrates the assignment of participant teams to groups.

Table 4.1 Assignment of participant teams to groups

System	Team Members	Pages	Requirements	Group
Color coord	3	13	22	HET
GesConnect	3	17	12	HET
PlayMaker	3	31	25	HET
PoliceVideo	3	15	5	HET
Harmedia	3	11	8	HET

CalPal	3	20	14	RET
Coupon Catcher	4	11	11	RET
EnterntainMe	3	17	20	RET
WhatsKitchen	3	14	12	RET
MansBestFriend	3	11	7	RET

4.3.4 Experiment Procedure

Figure 4.2 provides an overview of the study procedure, which included one training session and five experimental steps.

RET or HET Training: We held two training sessions, one for the members of the RET group and one for members of the HET group. In each of these 90-minute sessions, we trained participants on requirements inspections, fault detection, fault classes, and on how to use either the RET or HET (depending upon their group) to abstract and classify requirements errors from faults. We explained the error abstraction process in detail along with the RET or HET error classes. Finally, we trained the participants on how to use the abstracted errors to guide the reinspection of a requirements document.

Step 1 - Error Abstraction and Classification: This step served as a pretest to measure how well the participants understood human errors based on their ability to correctly abstract faults into errors and classify those errors. We gave the participants the SRS for the Parking Garage Control System (PGCS), which has been used in a number of studies. We also gave the participants a list of 10 of the 35 faults seeded in the PGCS SRS document. We chose these faults because they were used in a previous study and represent a cross-section of the error classes [3]. The participants used their knowledge from the RET or HET Training to abstract the faults into errors and classify those errors, into the respective taxonomies. Each participant performed this task independently. The output of this step was 31 PGCS Error Forms (15 from subjects who use the HET and 16 from subjects who used the RET).

Step 2 - Error-Based Inspection of PGCS: This step served as a second pretest to measure how well the participants understood human errors based on their ability to use error information to find additional requirements faults. Using the errors identified in Step 1 and their knowledge of the RET or HET, the participants individually inspected the PGCS SRS to identify any additional faults. If the participant identified a fault that was not related to one of the errors from Step 1, he/she abstracted that fault into its underlying error, added that error PGCS Error Form, and used that error to identify additional related faults. The output of this step was 31 PGCS Fault Forms (one per subject).

Step 3 - Development of SRS: In this step each team developed the SRS for their own respective systems.

Step 4 - Inspection of SRS: Participants individually used the RET or HET to inspect the SRSs developed by two other teams, one from the RET group and one from the HET group. The output of this step was 62 Individual SRS Fault Forms (one per subject).

Step 5 - Consolidate Fault Lists and Abstract Errors: Each team consolidated the results from Step 4 into one comprehensive fault list. The team used either the RET or HET to abstract those faults into the underlying errors. The output of this step was 10 Final Team Fault and Error Forms (one per team).

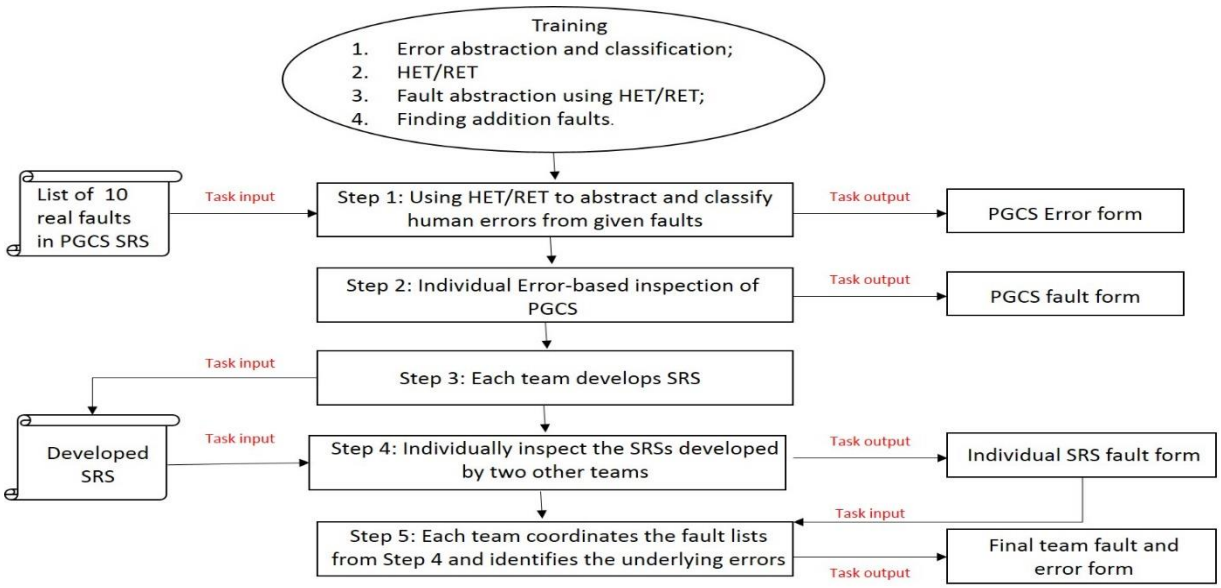


Figure 4.2 Experiment procedure

4.4. Results and analysis

This section provides a detailed analysis of the data collected during the study organized around the hypotheses posed in Section 4.3.1.

4.4.1 H1 - The better a developer understands human errors the less likely he/she will be to inject errors and faults into a requirements document

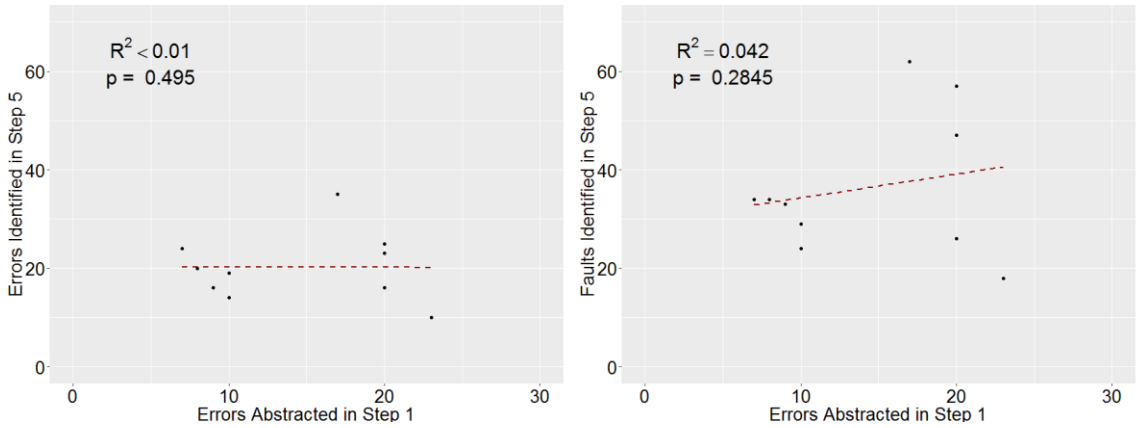
To test this hypothesis, we measured understanding in two ways: (1) ability to classify errors - how accurately participants abstracted faults into errors and classified those errors (Step 1) and (2) ability to use error information to find new faults - how effective the participants were at identifying additional faults based on the identified errors (Step 2). We measured likelihood of injecting errors and faults as the number faults in each SRS found by inspection (Step 4) and abstracted to errors by the teams (Step 5).

Effect of correctly abstracting and classifying errors (Step 1). Because the participants performed Step 1 independently but the SRSs were developed as a team, we computed the level of understanding for a team as the average of level of understanding of the

team members. Figure 4.3 plots the average percentage of errors correctly abstracted and classified in Step 1, by each team, against the number of unique errors and faults found during inspection (Step 4) and abstracted to errors (Step 5), for that team's SRS. Figure 4.3 also shows the results of the linear regression analysis between these pairs of variables. For this analysis, and the remainder in the paper, we used one-tailed significance tests because our hypotheses were for a negative correlation. The results of the linear regressions show a very slight, non-significant correlation in both cases. These results suggest that overall, a developer's ability to correctly abstract faults to errors and classify those errors does not seem to have an impact on the number of faults or errors he/she injects into his/her own SRS document.

Effect of finding additional faults (Step 2) For this analysis, we computed each team's level of understanding by counting the total number of unique faults identified by the team members during Step 2. Figure 4.4 plots that number against the number of unique faults found during inspection (Step 4) and abstracted to errors (Step 5) for the SRS developed by the team. Figure 4.4 also shows the results of the regression analysis between each pair of variables. In this case, both linear regression analyses show a strong, significant, negative correlation.

These results suggest that overall, a developer's increased ability to use error information (regardless whether it was learned in the context of the HET or the RET) to find faults during an error-based inspection is related to a decrease in the number of faults and errors he/she injects into his/her own SRS document. Yet, as shown in Figure 4.5, the ability to correctly abstract faults to errors and classify those errors does not seem to have an impact on the number of faults and errors injected into the requirements.

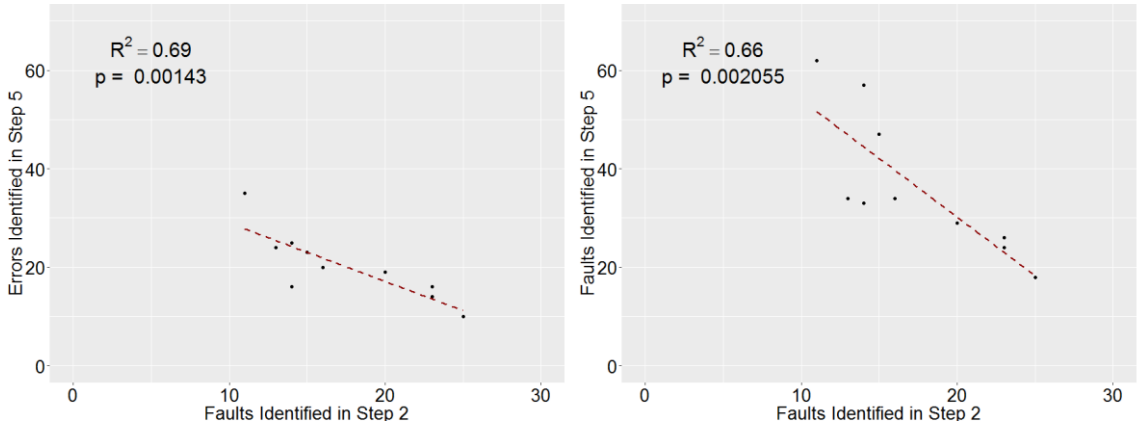


(a) Errors (b) Faults

Figure 4.3 Comparison between Step 1 and the number of faults/errors found in Step 5

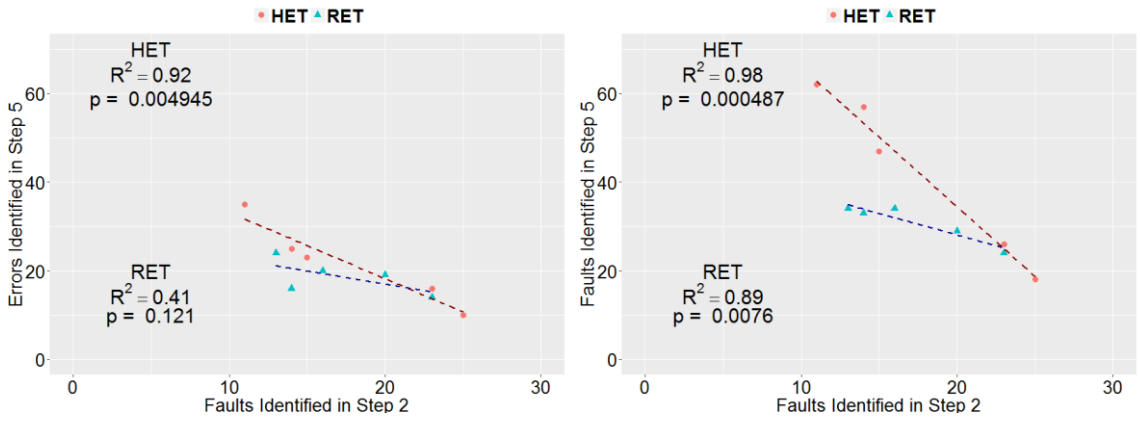
4.4.2 H2: Knowledge of the HET will provide more benefit for error/fault prevention than knowledge of the RET

Based on the results of H1, performance on the error-based inspection (Step 2) has a significant, negative correlation with the number of errors/faults inserted into the SRS. To compare the performance of the RET and the HET, we analyzed each approach separately.



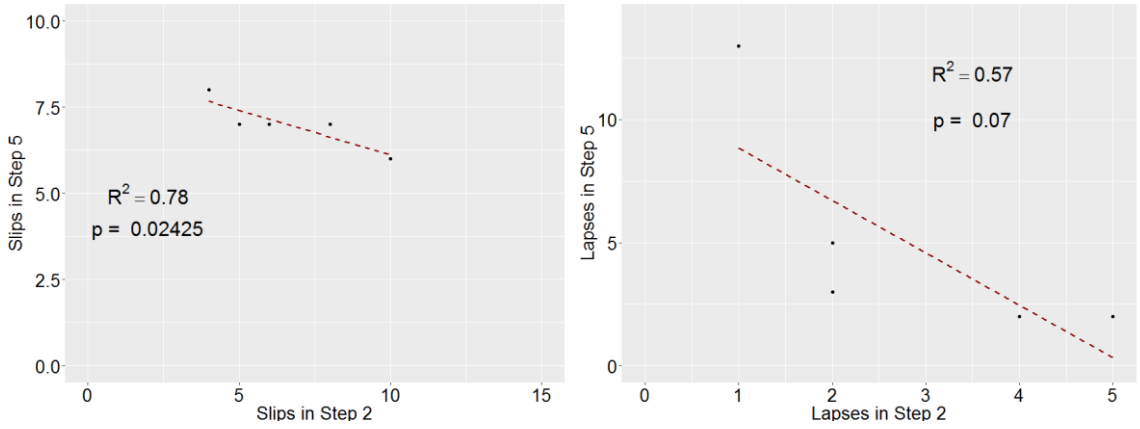
(a) Errors (b) Faults

Figure 4.4. Comparison between Step 2 test and the number of faults/errors found in Step 5

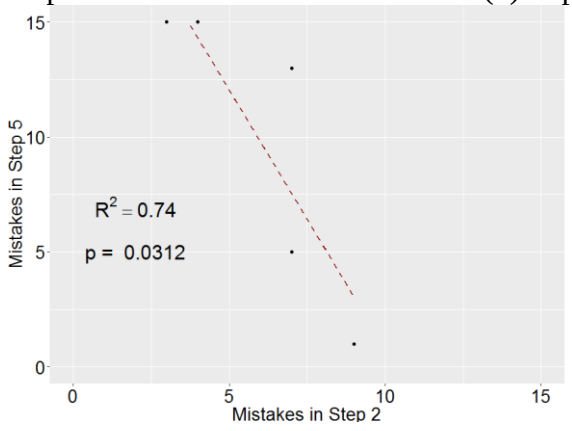


(a) Errors (b) Faults

Figure 4.5. Comparison of faults in Step 2 and faults/errors identified in Step 5



(a) Slips (b) Lapses



(c) Mistakes

Figure 4.6. HET Error Details

Figure 4.5 shows the results for each error taxonomy, along with the linear regression analysis. While teams in both groups exhibit a strong, negative correlation, the correlation for the HET teams is stronger. These results suggest that knowledge of the HET, which is based more closely on the concepts of human error, is more beneficial than knowledge of the RET.

4.4.3 H3: The better a developer understands each error type, the less likely he/she will be to insert errors/faults related to that type into a requirements document

The HET and the RET each organize errors differently at the top level. The RET uses an organization based on People, Process, and Documentation errors, developed bottom-up from the literature. The HET uses an organization based on Slips, Lapses, and Mistakes, based on a common taxonomy from cognitive psychology [22]. To better understand the results from H2, we performed

the same analysis, but this time separately for each of the high-level error classes. Figure 4.7 shows the results for the HET groups. Figure 4.7 shows the results for the RET groups. As before, the figures show the results of the regression analyses performed between the pairs of variables.

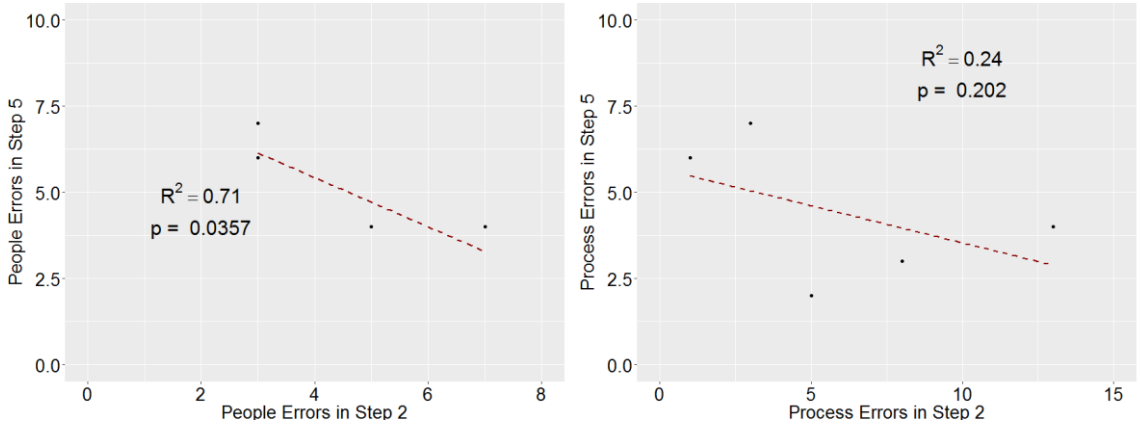
We can make a few observations about these results. First, for the HET, Slips and Mistakes both showed a significant negative correlation. This result is consistent with our previous studies [15]. Our previous work also found that Mistakes are the most common error type. Therefore, if this approach is able to help prevent those errors, it can be quite beneficial. Second, for the RET, the only error type that showed a strong, significant, negative correlation was People errors.

Based on the results from H2 (that HET is more helpful than RET), this result is not surprising. When comparing the RET to the HET, the People error type contains errors that are most similar to those contained in the HET.

4.5. Validation of Results with Industrial Data

The classroom study provided evidence that the knowledge of error information helps prevent developers from making similar errors while developing their own requirements documents. In addition to knowledge of the specific error types, we were interested in determining whether there were specific prevention mechanisms practitioners used to prevent errors from occurring. To achieve the goal of identifying prevention mechanisms and to provide guidance to industrial practitioners, we study the following additional research question.

RQ4: What prevention mitigation strategies can be used for each HET error type?



(a) People (b) Process

(c) Document

Figure 4.7. RET Error Details

To answer this research question, we gathered data from two industrial sources: interviews with CAPS developers and responses to the NaPiRE industrial survey.

4.5.1 Data Source

To investigate RQ4, I conducted interviews with professional developers in CAPS (a software development organization contained within the College of Engineering at UA) and analyzed data from the NaPiRE (*Naming the Pain in Requirements Engineering*) survey, a large-scale industrial survey conducted by collaborators to gather data regarding the sources of requirements problems. In the following sections, I describe each of these data sources in detail.

4.5.1.1 CAPS data

The first data source is the results of interviews I conducted with developers from the Center for Advanced Public Safety (CAPS) at the University of Alabama. Those developers were all involved in the requirements engineering process. As background, the CAPS organization builds different types of systems for different customers. The main systems that they developed are for traffic safety, homeland security, motor vehicles, analytics, health and human services, weather and disaster responses, emergency medical services and law enforcement. They also work with many Alabama state agencies as well as agencies in other states. In developing these system, CAPS has a core staff of business analysts, software engineers, and software developers work closely with clients to ensure that required features are implemented.

I chose CAPS developers as a data source due to their rich experience about software requirements development. Their experiences provide useful industry data to support my analysis. I interviewed seven developers who had experience in requirements development or requirements evaluation. During the interview, I first briefly introduced requirements errors and

their importance in the software development process. Then, I asked participants to describe any commonly occurring requirements errors they experienced on their projects along with any prevention mechanisms they employed or planned to employ. Next, I introduced the HET with a detailed explanation of each high-level error types and low-level error classes. Finally, I asked the interviewee whether any of the errors he/she described could be classified into the HET.

4.5.1.2 NaPiRE dataset

The second data source is the data from the NaPiRE survey. The main idea of NaPiRE is to conduct a broad survey investigating the status quo of requirements engineering in practice along with the problems requirements practitioners face. This survey is conducted bi-annually with practitioners across the world.

The data used here is drawn from respondents in 10 countries representing 226 companies (on participant per company). The survey presented 21 common requirements problems and asked the participants to choose the top 5 based on their experiences. For each of the 5 selected problems, the respondent listed the possible causes of the problem and the mitigation approaches they have taken or plan to take to address that problem [19].

As the topic of this survey and the data gathered are highly relevant to our work, i.e. providing information about the types of problems faced by requirements engineers and the sources of those problems, we have begun collaborating with the NaPiRE team. Here, I use the NaPiRE data to do a deep analysis to determine how well the HET describes the sources of requirements problems and to validate whether the HET is complete with regards to these real problems.

4.5.2 Data collection procedure

As described in Section 3.6.2, three reviewers analyzed the requirements errors and corresponding prevention mechanisms described by the developers in the interview and survey datasets. Based on the error descriptions and the prevention mechanisms, the three reviewers categorized these errors into the three high-level error types (slips, lapses and mistakes). Then we categorized the errors into the low-level error types. After performing this mapping, we had the list of error prevention mechanisms associated with each low-level error type. As a note, for this process, we used the updated HET described in Article 2. More details about the mapping process can be found in Section 3.6.2 in Article 2.

4.5.3 Data analysis results

To analyze the prevention mechanisms for each error type, we first collected all the prevention mechanisms for each error type from both datasets. Then we eliminated redundant or similar responses. Finally, we developed a list of mitigation strategies for each error type. Based on our understanding of the error type and the respondents' description of the mitigation strategy, we observed some cases where the mitigation approach did not appear to address the error to which it was associated. In those cases, we exclude the mitigation strategy from our analysis because its use would not reduce the likelihood of committing the underlying error. Table 4.2 shows the detailed information about the prevention mechanisms for each error types.

Table 4.2 Prevention Mechanisms for Error Types

Error Classes	Error Types	Prevention Mitigation (data come from)
Slips	Clerical Errors	Go over the dataset with the clients and users to make sure everything is correct prior to creating the requirements document (CAPS)
	Term Substitution	
Lapses	Loss of information from stakeholders	(1) Make a good communication with stakeholders (NaPiRE)

		(2) Designate a team member to take detailed notes during all meetings with the client (NaPiRE)
	Accidentally overlooking requirements	
	Multiple terms for the same concept errors	
Mistakes	Application errors: knowledge-based plan is incomplete.	(1) Evaluation of completed projects in order to derive lessons learned (NaPiRE) (2) Full requirements gathering along with wireframes supplied to developers (CAPS)
	Environment errors: knowledge-based plan is incomplete.	(1) Definition of a common structure to describe and explain requirements (NaPiRE) (2) Introduction and use of check lists for monitoring requirements along their life cycles (NaPiRE)
	Solution Choice errors	
	Syntax errors	
	Information Management errors: knowledge-based plan is incomplete.	(1) Planning and execution of trainings and specification workshops (in order to improve skill and performance) (NaPiRE) (2) Acquisition of (external) requirements experts (NaPiRE) (3) Make clear to focus on requirements and not on solution (NaPiRE) (4) Implementation of change management process (NaPiRE) (5) Creation of requirements specification template (NaPiRE) (6) Use of stronger formal reviews (NaPiRE) (7) Introduction of an artifact based quality management (and traceability) approach (NaPiRE)
	Wrong Assumption errors: knowledge-based plan is wrong	(1) Identification of process gaps; (CAPS) (2) Increase awareness to focus development on customer requirements (NaPiRE)
	Mistaken belief that it is impossible to specify non-functional requirements: knowledge-based plan is incomplete	(1) Implementation of measurement techniques for non-functional requirements (NaPiRE) (2) Definition of quality criteria to make requirements testable and measurable (NaPiRE)
	Not having a clear distinction between client and users	

	Lack of awareness of requirements sources: knowledge-based plan is incomplete	
	Inappropriate communication based on incomplete/faulty understanding of roles: knowledge-based plan is wrong	(1) Motivating project team (NaPiRE) (2) Involvement of production team (NaPiRE) (3) Introduction of a leader / manager for the delivery team (NaPiRE) (4) Planning and execution of regular communication events/ meetings (NaPiRE) (5) Promotion of knowledge transfer within project team (NaPiRE) (6) Planning and execution of trainings (in order to improve skill and performance) (NaPiRE) (7) Cross checks with solution designs (CAPS) (8) Definition of a standard or common structure to describe and explain requirements (NaPiRE) (9) Planning of elicitation before project begins (NaPiRE) (10) Introduction of an early feedback loop with customer (NaPiRE)
	Inadequate Requirements Process	(1) Introduction of an early feedback loop with development (NaPiRE) (2) Use of stronger formal reviews (NaPiRE) (3) Introduction of a standard (NaPiRE) (4) Implementation of change management process (NaPiRE) (5) Introduction and use of check lists for monitoring requirements along their life cycles (NaPiRE) (6) Implementation of a monitoring approach to ensure the coverage of user expectations / requirements (NaPiRE) (7) Implementation of a release process to ensure that requirements are final (NaPiRE) (8) Integrate Testing and RE (NaPiRE) (9) Introduction and use of a requirements quantification approach (NaPiRE) (10) Specific Prototyping phases and guaranteed grant/project time for starting over afterward these phases (CAPS) (11) Planning and execution of regular communication events/ meetings (NaPiRE) (12) Acquisition of (external) requirements experts (NaPiRE) (13) Planning and execution of trainings (in order to improve skill and performance) (NaPiRE) (14) Refinement of stakeholder analysis (CAPS)

		(15) Collecting the correct end users in the requirements meetings (CAPS) (16) Elaborating dependencies (NaPiRE) (17) Introduction of customer approvals (CAPS) (18) Creation of glossaries (NaPiRE) (19) Documentation of models and solutions (NaPiRE)
	Not defining the clear roles	(1) Definition of clear roles and responsibilities (NaPiRE) (2) Empower development team (CAPS) (3) Training/Coaching/Consulting (CAPS)
	Not properly allocating resources	(1) Introduction of an artifact based quality management (and traceability) approach (NaPiRE) (2) Planning and execution of trainings (in order to improve skill and performance) (NaPiRE) (3) Prioritization of activities / goals (NaPiRE) (4) Acquisition of (external) requirements experts (NaPiRE) (5) Definition and monitoring of measures (NaPiRE) (6) Evaluation and introduction of tools (NaPiRE) (7) Promotion of knowledge tansfer within project team (NaPiRE) (8) Sign-offs before implementation (CAPS)
	Not providing proper training	(1) Evaluation of completed projects in order to derive lessons learned (NaPiRE) (2) Evaluation and introduction of tools (NaPiRE) (3) Introduction of an artifact based quality management (and traceability) approach (NaPiRE) (4) Planning and execution of trainings (in order to improve skill and performance) (NaPiRE) (5) Use of stronger formal reviews (NaPiRE)
	Not providing changing management	(1) Implementation of change management process (NaPiRE)
	Failing to build team capabilities	Evaluation of completed projects in order to derive lessons learned (NaPiRE)
	Failed to provide tools and methods appropriate to project	(1) Introduction of an agile methodology (CAPS) (2) Introduction of communications tools (CAPS)
	Failed to recruite proper RE (Research Engineering) team members	(1) Documentation of models and solutions (NaPiRE) (2) Introduction of a quality assurance approach for specifications (NaPiRE) (3) Planning and execution of regular communication events/ meetings (NaPiRE) (4) Planning and execution of trainings (in order to improve skill and performance)

		(NaPiRE) (5) Integrate Testing and RE (NaPiRE)

4.5.4 Analysis

Based on Table 4.2, we can find that for each of error type that have happened in the real industry, developers have taken or plan to take a bunch of mitigation to prevent the related errors happen. Meanwhile, we can find that based on these different types of mitigation that provided by different participants, they have some kinds of similar characteristics.

4.5.4.1 Detail prevention mitigation for each error type

In the following, we analyzed the detail characteristics of prevention mitigation for each corresponding error type.

- **Clerical Error --** According to the description of clerical error, it mainly happens because of the carelessness of developers while transferring information from one format to another, thus, the core idea of mitigation is to read the requirements information in detail and to make sure that the information that developers transferred is correct.
- **Loss of information from stakeholders --** The main reason that cause this error happen is that RE forget or discard the information that provided by the stakeholders, so to prevent this kind of error mainly focused on two aspects: have good communication and record all the meeting notes.
- **Application errors --** This error occurs primarily due to misunderstandings of the requirements engineers. Therefore, developers address this error by evaluating and understanding the entire set of requirements and the projects, and by deriving lessons from this extraction process to help them fully understand the projects and avoid misunderstandings.

- **Environment errors --** This error results from the lack of knowledge about the available infrastructure. To avoid this kind of error, developers think it is useful to have a standard or common structure for requirements description and monitoring.

- **Information Management errors –** This error results from a lack of knowledge about standard requirements process or documentation practice. Developers provided two kinds of prevention mechanisms: Improving the skill of the requirements engineer and implementing a standard template or management process.

- **Wrong Assumption errors --** Developers' misassumption for system features or client requirements causes this error. To avoid it, two measurements were provided: definitely follow what the customer's required and make the requirements consistent.

- **Mistaken belief that it is impossible to specify non-functional requirements --** Developers' mistaken belief about the non-functional requirements leads them to omit the necessary process to specify these requirements. To avoid this error, it is necessary to use techniques to help developers specify non-functional requirements.

- **Inappropriate communication based on incomplete/faulty understanding of roles –** This error is caused by a lack of understanding of requirements engineering roles. The main approach for mitigation is to improve the skills and communication of the requirements engineers.

- **Inadequate Requirements Process –** This error occurs because Requirements engineers do not follow all the required requirement processes. The main mitigation approach is to improve the requirement engineers' knowledge and introduce necessary requirements process.

- **Management errors --** Management error result from the lack of a management plan. There are several kinds of management errors can occur during the requirements stage,

including: failing to define clear roles, not allocating resources properly, and not providing enough training for the management team. The main prevention approach that developers provided use are: (1) provide training or lessons to improve the team skills; (2) introduce proper tools or methods; (3) implement the standard development process.

4.5.4.2 Same mitigation for different error types

During the analysis, we observed the same prevention mechanism mentioned to address different error types. One possible source of this commonality comes from the fact that we can divide Mistake errors into two classes, internal errors and interface errors (See Section 3.6.3 in Article 2 for more details). Internal are primarily caused by lack of knowledge or incorrect assumptions by the requirements engineering team. Because each of these error is relatively unique, the prevention mechanisms must differ for each one. Conversely, interface errors, which are primarily caused by poor communication with clients or by lack of adequate management can be prevented by improving management of the RE team or by additional training.

For example, the mitigation "Planning and execution of trainings and specification workshops (in order to improve skill and performance)" was provided by same person as the mitigation strategy for five error types: Information management errors, inappropriate communication based on incomplete/faulty understanding of roles, not properly allocating resources, inadequate requirements process, and management errors. These errors are all interface errors. Therefore, the same mitigation approach can be used. We also found cases where the same mitigation approach was mentioned by different people for different errors. For example, Acquisition of (external) requirements experts can be used to mitigate: information management errors, inadequate requirements process and management errors.

This finding shows that employing some prevention mitigation will enable teams to prevent multiple types of errors. This result also provides the insight that same mitigation can be used for different kinds of error types. Therefore, when developers know the related prevention mitigation, they can prevent many kinds of error types and reduce the likelihood of injecting the errors while developing their own SRS and improve the whole system quality.

4.6. Threats to validity

This section describes the primary threats, along with the steps taken to mitigate them where possible.

4.6.1 Threats to validity of classroom study

Internal Validity First, even though we have no evidence to the contrary, we have no way of ensuring that the participants followed the specified HET or RET processes. Therefore, this threat is minimal. In Step 4, we did have participants inspect the SRS documents from two other groups. Second, it is possible that by the second inspection they became fatigued and did not perform as well. To mitigate this threat, we gave them enough time to perform both inspections.

External Validity The participants were undergraduate students. Therefore, the results are not directly applicable in an industrial context. Even so, the students were building a real system, so the activities performed in this study did have relevance to the projects. We will need additional studies to understand how these results apply in industry.

Construct Validity First, we defined understanding of human error in two ways in Section 4.1. Based on our study design, these definitions seemed to be the most appropriate. It is possible that other definitions would have provided different results. Second, because the students were building their own systems (rather than using systems with seeded faults/errors) we do not know the total number of errors made during SRS development. Our conclusions are

based only on the errors that reviewers identified in Step 4. Third, the students were working on different projects and it is hard to compare the results.

4.6.2 Threats to validity of industry studies

Internal Validity In the process of collecting the prevention mechanisms for the requirements errors, all the information is subjective. We did not perform any objective studies to determine whether the responses are truly valid. Therefore, we need more studies to validate the usefulness of the prevention approaches.

Construct validity In this study, we just asked participants to provide the mitigation for the requirements errors that they confronted based on their past experience and memory. For the survey, the respondents were not aware of the HET. Therefore, the respondents may have omitted human errors that did not immediately come to mind.

4.7. Conclusion and future work

From the classroom study, the better the students understood human errors from the training process the fewer errors they made in their own documents. Therefore, we can conclude that knowledge of the HET is beneficial for the students. For both the classroom data and the industry data, *Mistake* errors are the most prevalent. Furthermore, comparing the HET and the RET in the classroom study showed that knowledge of the HET had a greater defect prevention effect than knowledge of the RET. The results of the industrial studies revealed some specific error prevention mechanisms. These results showed that improving communication with stakeholders, improving management, and improving skills and knowledge could have a great effect on error prevention.

The primary contribution of this paper are (1) conclusions about the type of human error knowledge that helps prevent errors and faults during software development, (2) evidence that a taxonomy based directly on human error information (the HET) is more effective in fault

prevention, (3) insight into how the specific error types in each taxonomy contribute to the overall result, and (4) provide the detail prevention mitigation for each error types and provide guidelines for future developers and researchers.

As future work, we intend to perform more validation of the HET for error and fault prevention. While the results of the classroom study showed a positive impact of knowledge of the HET, we need to perform similar studies in professional settings to validate whether these results hold with participants who are more experienced. In the studies that we conducted with industrial professionals, we did not ask them to use the HET, only to report errors they had encountered. To address this shortcoming in our current work, we will conduct further, controlled studies with industrial practitioners to understand the effects of the HET and identify any aspects that it does not currently address. In this study, as we have done in classroom setting, we will firstly do training for the professional participants with HET/RET, then ask them to develop a real system with the knowledge of HET/RET. Then we can analyze whether the results obtained in the industrial setting are consistent with those obtained in the classroom setting.

References

[1] ISO, IEC and IEEE. Systems and software engineering: vocabulary. ISO/IEC/IEEE 24765:2010(E), 2010: 1- 418.

[2] Anu, V., Hu, W., Carver, J.C., Walia, G.S., Bradshaw, G.: Development of a human error taxonomy for software requirements: A systematic literature review. Technical Report NDSU-CS-TR-16-001, North Dakota State University (2016), http://vaibhavanu.com/NDSU-CS/TR-16-001.pdf

[3] Anu, V.K., Walia, G.S., Hu, W., Carver, J.C., Bradshaw, G.: Effectiveness of human error taxonomy during requirements inspection: An empirical investigation. In: 2016 Int'l Conf. on Soft. Eng. and Knowledge Eng, 2016: 531-536.

[4] Card, D.N.: Learning from our mistakes with defect causal analysis. IEEE software, 1998, 15(1): 56-63.

[5] Carver, J., Nagappan, N., Page, A.: The impact of educational background on the effectiveness of requirements inspections: An empirical study. IEEE Trans. on Soft. Eng., 2008, 34(6): 800-812.

[6] Chen, J.C., Huang, S.J.: An empirical analysis of the impact of software development problem factors on software maintainability. J. of Sys. and Soft., 2009, 82(6): 981-992.

[7] Chillarege, R., Bhandari, I.S., Chaar, J.K., Halliday, M.J., Moebus, D.S., Ray, B.K., Wong, M.Y.: Orthogonal defect classification-a concept for in-process measurements. IEEE Transactions on software Engineering, 1992, 18(11): 943-956.

[8] Dethomas, A.: Technology requirements of integrated, critical digital fight systems. In: Guidance, Navigation and Control Conf., 1987: 2602.

[9] Diller, T., Helmrich, G., Dunning, S., Cox, S., Buchanan, A., Shappell, S.: The human factors analysis classification system (hfacs) applied to health care. American Journal of Medical Quality, 2013: 181-190.

[10] Freimut, B., Denger, C., Ketterer, M.: An industrial case study of implementing and validating defect classi_cation for process improvement and quality management. In: 11th IEEE Int'l Software Metrics Symposium, 2005.

[11] Grady, R.B.: Software failure analysis for high-return process improvement decisions. Hewlett Packard Journal, 1996, 47: 15-24.

[12] Graham, M.: Software defect prevention using orthogonal defect prevention. 2005.

[13] Hamill, M., Goseva-Popstojanova, K.: Common trends in software fault and failure data. IEEE Transactions on Software Engineering, 2009, 35(4): 484-496.

[14] Hayes, J.H.: Building a requirement fault taxonomy: Experiences from a nasa verification and validation research project. In: 14th Int'l Symposium on Software Reliability Engineering, 2003: 49-59.

[15] Hu, W., Carver, J.C., Anu, V., Walia, G., Bradshaw, G.: Detection of requirement errors and faults via a human error taxonomy: A feasibility study. Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. ACM, 2016.

[16] Kumaresh, S., Baskaran, R.: Experimental design on defect analysis in software process improvement. In: 2012 Int'l Conf. on Recent Advances in Computing and Software Systems (RACSS), 2012: 293-298.

[17] Lanubile, F., Shull, F., Basili, V.R.: Experimenting with error abstraction in requirements documents. In: SoftwareMetrics Symposium, 1998. Metrics 1998. Proc. Fifth Int'l, 1998: 114-121.

[18] Leszak, M., Perry, D.E., Stoll, D.: A case study in root cause defect analysis. In: Proc. of the 22nd Int'l Conf. on Soft. Eng. ACM, 2000: 428-437.

[19] Masuck, C.: Incorporating a fault categorization and analysis process in the software build cycle. Journal of Computing Sciences in Colleges, 2005, 20(5): 239-248.

[20] Mays, R.G., Jones, C.L., Holloway, G.J., Studinski, D.P.: Experiences with defect prevention. IBM Systems Journal, 1990, 29(1): 4-32.

[21] Pooley, R., Senior, D., Christie, D.: Collecting and analyzing web-based project metrics. IEEE software, 2002, 19(1): 52.

CHAPTER 5

DISSERTATION CONCLUSION

The software development process is a human-centric process, especially during the requirements stage. The elicitation and documentation of software requirements are thoroughly human-based activities. As a result, when problems occur, they are likely the result of failures in the human mental processing faculties. Cognitive Psychologists have long studied the topic of Human Error, focusing on how the human metal processes fail when carrying out various tasks. Thus, the goal of my dissertation is to use insights from cognitive psychology research on human errors to develop a human error taxonomy (HET) and evaluate whether the HET is useful for detecting and preventing faults and errors. The dissertation contains three articles that achieve this goal.

Article 1 describes a systematic literature review that uses the cognitive theory of human errors to develop a formalized taxonomy of human errors that occur during the requirements phase of the software lifecycle. In this literature review, we followed the traditional systematic literature review (SLR) approach and searched published papers from 2006 to 2014 related to software requirements errors and human cognitive errors. From the 38 included papers, we collected the human errors that occur during software requirements engineering. Using that information, we constructed a structured human error taxonomy (HET) to organize this information into a useful structure.

Article 2 describes a series of studies to evaluate the feasibility and usefulness of the HET for fault and error detection. In this article, we performed two types of studies. The first one

is a feasibility study with senior-level undergraduate students. The second is an industrial study we conducted with professional developers. In the feasibility study, students used the HET to detect errors in their own requirements document and to find additional faults during a reinspection of their own requirements document. These participants had a positive impression of the HET. In the industrial study, respondents used their professional experience as background to describe the types of errors that occur during their requirements engineering tasks. Using that information, we classified the reported errors using the HET to determine whether it was capable of describing the problems that actually occur in practice. The results showed that, while the HET could describe most of the reported errors, some HET error types were absent and we had to add a new error type to cover errors reported in industry. As a result of this study, we improved the HET to be more representative of practice.

Article 3 describes a series of studies used to evaluate the usefulness of error information in general and the HET specifically for preventing errors. In this article, we also performed two types of studies. The first is a control group study with senior-level undergraduate students. The second is the same industrial study from Article 2 (although we use different data). In the control group study, we found that increased knowledge of the human error information helped prevent students from making related errors in their own requirements documents. In the industrial study, we were able to classify the prevention mechanisms industrial professionals use to combat the errors in the HET. Through this analysis, we find that the same prevention mechanisms can be used to address different requirement errors.

The primary contributions of this dissertation are:

- Illustration of the ability and value of applying human error research to a software engineering problem through close interaction between software engineering experts and cognitive psychology experts;
- Development a human error taxonomy to organize the human errors made during requirements into a systematic framework;
- Validation of the usefulness of the HET for detecting human errors; and
- Evaluation of the usefulness of the HET for preventing human errors and the related prevention measurement.

APPENDIX A

This appendix describes the search strings along with their search focus (Table XI) that were executed on IEEExplore, INSPEC, ACM Digital Library, SCIRUS (Elsevier), Google Scholar, PsychINFO (EBSCO), and Science Citation Index databases.

Table XI. Search strings

String#	Search Focus	Detailed Search string
1	Software quality improvement approach	((software OR development OR application OR product OR project) AND (quality OR condition OR character OR property OR attribute OR aspect) AND (improvement OR enhancement OR advancement OR upgrading OR ameliorate OR betterment) AND (approach OR process OR system OR technique OR methodology OR procedure OR mechanism OR plan OR pattern))
2	Software inspection methods	((software OR development OR application OR product OR project) AND (inspection OR assessment OR evaluation OR examination OR review OR measurement) AND (approach OR process OR system OR technique OR methodology OR procedure OR mechanism OR plan OR pattern))
3	Error abstraction OR root causes	(error OR mistake OR problem OR reason OR fault OR defect OR imperfection OR flaw OR lapse OR slip OR err) AND (abstraction OR root cause OR cause))
4	Requirement stage errors	((requirement OR specification) AND (phase OR stage OR situation OR division OR period OR episode OR part OR state OR facet) AND (error OR mistake OR problem OR reason OR fault OR defect OR imperfection OR flaw OR lapse OR slip OR err))
5	Software error/fault/defect taxonomy	((software OR development OR application OR product OR project) AND (error OR mistake OR problem OR reason OR fault OR defect OR imperfection OR flaw OR lapse OR slip OR err) AND (taxonomy OR classification OR categorization OR grouping OR organization OR terminology OR systematization))
6	Human error classification	((human OR cognitive OR individual OR psychological) AND (error OR mistake OR problem OR reason OR fault OR defect OR imperfection OR flaw OR lapse OR slip OR err) AND (taxonomy OR classification OR categorization OR grouping OR organization OR terminology OR systematization))

APPENDIX B

This appendix describes the distribution of primary studies across different journal and conference avenues.

Table XII. Distribution of primary studies across venues

Source	Study Count
International Journal of Computer Applications	1
IEEE International Conference on Engineering of Complex Computer Systems (ICECCS)	1
Information Sciences Journal	1
IEEE Frontiers in Education Conference	1
International Conference on Software Engineering Advances	1
ACM-IEEE international symposium on Empirical software engineering and measurement (ESEM)	1
Workshop em Engenharia de Requisitos (WER)	1
IEEE/NASA Software Engineering Workshop (SEW)	1
Minds and Machines Journal	1
Malaysian Conference in Software Engineering (MySEC)	1
Information Systems Journal	2
Information and Software Technology Journal	1
International Workshop on Rapid Continuous Software Engineering (RCoSE)	1
International Conference on Recent Advances in Computing and Software Systems (RACSS)	1
ISSAT International Conference on Reliability and Quality in Design	1
IEEE International Requirements Engineering Conference (RE)	1
IEEE Annual Computer Software and Applications Conference Workshops	1
IEEE International Conference on Cognitive Informatics	1
IFIP/IEEE International Symposium on Integrated Network Management (IM)	1
IEEE International Symposium on Software Reliability Engineering (ISSRE),	1
International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS)	1
International Conference on Software Engineering and Data Mining (SEDM),	1
Iberian Conference on Information Systems and Technologies (CISTI)	1
International Conference on Software Engineering and Knowledge Engineering (SEKE)	1
Journal of Object Technology	1

Journal of Systems and Software	2
The EDP Audit, Control, and Security Newsletter (Journal)	1
International Management Review Journal	1
American Journal of Medical Quality	1
IEEE International Conference on Automation and Logistics (ICAL)	1
Safety Science Journal	1
IFIP Advances in Information and Communication Technology (Journal)	1
International conference on Computer Safety, Reliability, and Security (SAFECOMP)	1
International Conference on Industrial Engineering and Engineering Management	1
Studies in Health Technology and Informatics (Book Series)	1
Professional Safety Journal	1

APPENDIX C

This appendix provides a list of exclusively those 18 papers that provided input to the HET. Some of these papers are also cited in the main paper. Please note that the references marked with an asterisk (*) are also cited in the main paper and are included in the reference list for the main paper.

[1]. Basili VR, Perricone BT (1984) Software Errors and Complexity: An Empirical Investigation. Commun ACM 27:42–52. doi: 10.1145/69605.2085

[2]. Basili VR, Rombach HD (1987) Tailoring the Software Process to Project Goals and Environments. In: Ninth Int. Conf. Softw. Eng. IEEE Press, California, USA, pp 345–357

[3]. Bhandari I, Halliday MJ, Chaar J, Chillarege R, Jones K, Atkinson JS, Lepori-Costello C, Jasper PY, Tarver ED, Lewis CC, Yonezawa M (1994) In-process improvement through defect data interpretation. IBM Syst J 33:182–214. doi: 10.1147/sj.331.0182

[4]. Bjarnason E, Wnuk K, Regnell B (2011) Requirements are slipping through the gaps - A case study on causes & effects of communication gaps in large-scale software development. In: Proc. 2011 IEEE 19th Int. Requir. Eng. Conf. RE 2011. pp 37–46

[5]. Coughlan J, Macredie RD (2002) Effective Communication in Requirements Elicitation: A Comparison of Methodologies. Requir Eng 7:47–60. doi: 10.1007/s007660200004

[6]. *Firesmith D (2007) Common requirements problems, their negative consequences, and the industry best practices to help solve them. J Object Technol 6:17–33. doi: 10.5381/jot.2007.6.1.c2

[7]. Galliers J, Minocha S, Sutcliffe AG (1998) A causal model of human error for safety critical user interface design. ACM Trans Comput Interact 5:756–769.

[8]. *Huang F, Liu B, Huang B (2012) A Taxonomy System to Identify Human Error Causes for Software Defects. Proc. 18th Int. Conf. Reliab. Qual. Des. ISSAT 2012

[9]. *Kumaresh S, Baskaran R (2010) Defect Analysis and Prevention for Software Process Quality Improvement. Int J Comput Appl 8:42–47. doi: 10.5120/1218-1759

[10]. *Kushwaha DS, Misra AK (2006) Cognitive software development process and associated metrics - A framework. In: Proc. 5th IEEE Int. Conf. Cogn. Informatics, ICCI 2006. pp

255–260

[11]. *Lehtinen TOA, Mantyla MV., Vanhanen J, Itkonen J, Lassenius C (2014) Perceived causes of software project failures - An analysis of their relationships. Inf Softw Technol 56:623–643. doi: 10.1016/j.infsof.2014.01.015

[12]. *Leszak M, Perry DE, Stoll D (2000) A case study in root cause defect analysis. Proc 22nd Int Conf Softw Eng - ICSE ’00 428–437. doi: 10.1145/337180.337232

[13]. *Lopes M, Forster C (2013) Application of human error theories for the process improvement of Requirements Engineering. Inf Sci (Ny) 250:142–161.

[14]. Lutz RR (1993) Analyzing software requirements errors in safety-critical, embedded systems. Proc IEEE Int Symp Requir Eng 126–133. doi: 10.1109/ISRE.1993.324825

[15]. Mays RG, Jones CL, Holloway GJ, Studinski DP (1990) Experiences with Defect Prevention. IBM Syst J 29:4–32. doi: 10.1147/sj.291.0004

[16]. Nakashima T, Oyama M, Hisada H, Ishii N (1999) Analysis of software bug causes and its prevention. Inf Softw Technol 41:1059–1068. doi: 10.1016/S0950-5849(99)00049-X

[17]. De Oliveira KM, Zlot F, Rocha AR, Travassos GH, Galotta C, De Menezes CS (2004) Domain-oriented software development environment. J Syst Softw 72:145–161. doi: 10.1016/S0164-1212(03)00233-4

[18]. Zhang X, Pham H (2000) An analysis of factors affecting software reliability. J Syst Softw 50:43–56. doi: 10.1016/S0164-1212(99)00075-8

APPENDIX D IRB CERTIFICATE

This appendix includes all the copy of the Institutional Review Board (IRB) certification.

Office for Research

Institutional Review Board for the Protection of Human Subjects

THE UNIVERSITY OF ALABAMA RESEARCH

September 29, 2015

Wenhua Hu
Dept of Computer Science
College of Engineering
Box 870290

Re: IRB # 15-OR-294, "Evaluating the Use of Requirements Error Information"

Dear Mr. Hu:

The University of Alabama Institutional Review Board has granted approval for your proposed research.

Your application has been given expedited approval according to 45 CFR part 46. Approval has been given under expedited review category 7 as outlined below:

(7) Research on individual or group characteristics or behavior (including, but not limited to, research on perception, cognition, motivation, identity, language, communication, cultural beliefs or practices, and social behavior) or research employing survey, interview, oral history, focus group, program evaluation, human factors evaluation, or quality assurance methodologies.

Your application will expire on September 28, 2016. If your research will continue beyond this date, please complete the relevant portions of the IRB Renewal Application. If you wish to modify the application, please complete the Modification of an Approved Protocol Form. Changes in this study cannot be initiated without IRB approval, except when necessary to eliminate apparent immediate hazards to participants. When the study closes, please complete the Request for Study Closure Form.

Please use reproductions of the IRB approved stamped consent forms to obtain consent from your participants.

Should you need to submit any further correspondence regarding this proposal, please include the above application number.

Good luck with your research.

Sincerely,

[redacted]

358 Rose Administration Building
Box 870127
Tuscaloosa, Alabama 35487-0127
(205) 348-8461
FAX (205) 348-7189
TOLL FREE (877) 820-3066

IRB Project #: 16-OR-261

UNIVERSITY OF ALABAMA
INSTITUTIONAL REVIEW BOARD FOR THE PROTECTION OF HUMAN SUBJECTS
REQUEST FOR APPROVAL OF RESEARCH INVOLVING HUMAN SUBJECTS

I. Identifying information

	Principal Investigator	Second Investigator	Third Investigator
Names:	Wenhua Hu	Jeffery Carver	
Department:	Computer Science	Computer Science	
College:	Engineering	Engineering	
University:	Alabama	Alabama	
Address:	Box 860290	Box 870290	
Telephone:	205-856-1378	8-9829	
FAX:	8-0215	8-0219	
E-mail:	whu10@crimson.ua.edu	carver@cs.ua.edu	

Titleof Research Project: Evaluating the Use o'Human Error Taxonomy

DateSubmitted: 7/15/16
Funding Source: National Science Foundation

Type of Proposal ☒ New ☐ Revision ☐ Renewal Please attach a renewal application ☐ Completed ☐ Exempt
Please attach a continuing review of studies form
Please enter the original IRB # at the top of the page

UA faculty or staff member signature: [redacted]

II. NOTIFICATION OF IRB ACTION (to be completed by IRB):
Type of Review: ___ Full board X Expedited

IRB action:
___ Rejected Date: ___
___ Tabled Pending Revisions Date: ___
___ Approved Pending Revisions Date: ___
X Approved-this proposal complies with University and federalregulations for the protection of human subjects.
Approval is effective until the following date: 7/26/2017
Items approved: X Research protoco (dated ___)
X Informed consent (dated ___)
___ Recruitment materials (dated ___)
___ (dated ___)

Approval signatur [redacted] Date 07/27/16