USING HUMAN ERROR MODELS TO IMPROVE THE QUALITY OF SOFTWARE REQUIREMENTS

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

By

Vaibhav Kumar Anu

In Partial Fulfillment of the Requirements
for the Degree of
DOCTOR OF PHILOSOPHY

Major Department:
Computer Science

May 2018

Fargo, North Dakota

North Dakota State University
Graduate School

Title

USING HUMAN ERROR MODELS TO IMPROVE THE QUALITY OF SOFTWARE REQUIREMENTS

By

Vaibhav Kumar Anu

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

DOCTOR OF PHILOSOPHY

SUPERVISORY COMMITTEE:

Dr. Gursimran Singh Walia
Chair

Dr. Kendall Nygard

Dr. Jun Kong

Dr. Verlin B. Hinsz

Approved:

June 6, 2018
Date

Dr. Kendall Nygard
Department Chair

ABSTRACT

Creating high quality software is a primary concern for software development organizations. Researchers have devoted considerable effort in developing quality improvement methods that help software engineers detect faults early in the development lifecycle (when the faults are cheapest to detect and repair). While useful, the available approaches still cannot make sure that Software developers are able to identify all or even a significantly large portion of faults. This is because they do not help software developers identify errors (i.e., underlying cause of faults) that may have led to the insertion of the faults (i.e., manifestation of error). This lack of focus on errors causes some faults to be overlooked which impacts quality of software produced.

Requirements engineering is the most people-intensive phase of software development. Thus, requirements engineering is more prone to human error when compared to other phases of software development. To that end, this dissertation focuses on understanding the human error causes of requirements faults. The central idea that drives this dissertation is that, knowledge of errors that commonly occur during the requirements engineering process can help software developers in detecting faults that are otherwise overlooked when using traditional approaches and also help them to avoid making errors when developing requirements.

Human error research focuses on understanding and classifying the fallibilities of human cognition. This dissertation combines requirements error information (gathered from Software Engineering literature) with the general accounts of human error and human error models (gathered from the Psychology literature). There are three steps to this work:

- Development of a requirements phase human error taxonomy,
- Empirical validation of the taxonomy's usefulness for understanding requirements faults and errors, and

- Development and subsequent validation of a formal software inspection technique based on the taxonomy.

As a result of this dissertation, a structured Human Error Taxonomy (HET) that classifies requirements phase errors was created with direct ties to the existing human error theories. Several empirical validations of the taxonomy have helped in:

- Successfully demonstrating the taxonomy's usefulness for understanding requirements faults and errors, and
- Developing a formal HET-based Error Abstraction and Inspection (EAI) approach and supplementary human error investigation tools.

ACKNOWLEDGEMENTS

The research that is presented in this dissertation is not a reflection of my abilities, but instead the immeasurable contributions from my advisors, peers, family, and friends. I am forever grateful for these contributions.

Particularly, Dr. Gursimran Singh Walia, who has been an invaluable voice in my research. Dr. Walia graciously agreed to be my PhD mentor and has steered my research in the right direction ever since I started working under his advisement. Dr. Walia has not only encouraged me to pursue my ideas, but by questioning my oftentimes poorly formulated concepts he has forced deeper contemplation. Words cannot express the immense gratitude that I feel for Dr. Walia's continuous support and guidance throughout my PhD.

I owe credit to Dr. Jeffrey C. Carver and Dr. Gary L. Bradshaw at University of Alabama-Tuscaloosa and Mississippi State University, respectively. My research has significantly benefitted from the timely advice they gave to refine my ideas, research questions, and research goals. I am grateful to have had Dr. Carver and Dr. Bradshaw as mentors because without their expert insights my dissertation would be lacking in impactful results.

I owe immense gratitude to the Department of Computer Science at North Dakota State University. Said plainly, this dissertation would not have been possible without the support provided by the department's faculty members and administrative support team. I also owe gratitude to the students at the department for participating in my research.

DEDICATION

For my parents, Neelam Tyagi and Komesh Kumar Tyagi.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

HET.....Human Error Taxonomy.

RET.....Requirements Error Taxonomy.

SE.....Software Engineering.

RE .....Requirements Engineering.

CS.....Computer Science.

EA .....Error Abstraction.

EAI.....Error Abstraction and Inspection.

SRS .....Software Requirements Specification.

FC.....Fault Checklist.

RQ.....Research Question.

SLR .....Systematic Literature Review.

1. INTRODUCTION

Software development is an extremely human-centric activity that involves participation of several people who perform various developmental tasks. During the early phase of software development, software developers elicit user needs, translate those needs into requirements, and validate the requirements for correctness, completeness, and other quality attributes. Owing to the involvement of various stakeholders (both technical and non-technical) in this process, there is the potential for *human errors* (i.e., human cognitive failures) to occur.

Cognitive Psychologists have long studied and analyzed the types of errors that people commit when performing different types of tasks in safety-critical, human-centric domains such as aviation, medicine, railway system, and nuclear power plants [1–4]. This research has not only improved the human performance in these domains, but has also lead to a decrease in unfavorable incidents by helping organizations identify and prevent errors. Because software development is a complex, human-centric activity, the fallibility of human cognition (during activities such as user-need elicitation, requirements and design analysis) leads to human errors. These human errors can then lead to various types of faults. In the same manner that human error research has benefited other domains by providing error identification/prevention mechanisms, this dissertation hypothesizes that properly applied human error research can have similar quality-improvement effects in software development. To that end, in this dissertation, I have applied the *Cognitive Psychology research on human errors* to propose a human error-based approach for improving software quality.

1.1. Historical Perspective on Software Quality Improvement

Software quality (or lack thereof), is a primary concern for both software developers and researchers. Much research and experience has shown that identifying and correcting problems

during the earlier phases of software development (i.e., requirements phase) can save significant project costs (associated with rework) [5–7]. Research has shown that about 40% of *total project budget* is spent on rework [5, 8], and furthermore, finding and fixing requirements faults consumes between 70% to 85% of the *total project rework cost* [9, 10]. To alleviate these requirements quality problems, researchers have proposed and empirically validated the usefulness of a variety of requirements quality improvement techniques ranging from fault-checklist based artifact reviews [11–14] to N-Fold inspections [15–17]to Perspective Based Reading [18–20]. The fault-checklist technique, which is the most popular requirements inspection method, uses the "lessons learned" from historical fault data to suggest ways for reviewers to identity faults [21, 22]. N-Fold inspections improves the fault-checklist technique by replicating the inspection activities using *N* independent inspection teams. Perspective Based Reading (PBR) technique focusses on examining requirements artifacts from different perspectives of the potential users of the artifact [18]. PBR tries to improve inspection efficiency by minimizing the overlap among the faults detected by the inspectors. Although successful in improving requirements quality, research has shown that even the most faithful application of inspection approaches like fault-checklist, N-fold inspections, and PBR can help locate only 50-60% of the faults present in requirements documents [13, 21].

As can be seen from the discussion above, even though the above-mentioned techniques (like PBR) have achieved varying degrees of success, they cannot help software development teams find and fix all requirements problems because they treat the symptoms of the problem (i.e., faults) and not the underlying causes of the problem (i.e., human error) [22, 23]. Identifying these human errors can help Software engineers understand why problems occurred, find and fix related faults, and prevent errors and faults from happening in the future. To that end, this

dissertation hypothesizes that focusing on human errors (i.e., the underlying causes of faults) as compared to focusing on faults alone can help software development teams find all or most of the faults, and hence provide a more complete and sound method to address the software quality problem.

1.2. Dissertation Goals

Human error research has been successfully adapted in various domains like aviation, medicine, and process control [4, 24–28] for improving both the process quality and the product quality. Human error research relies on studying human information processing models to investigate mental processes that lead to cognitive failures. These cognitive failures are also referred to as human errors or mental errors.

Faults/defects in software engineering artifacts arise during the process of translating (or processing) information gathered from the users into requirements, design, and then code. Each of these activities are human-centric and are prone to human cognitive failures (or human errors). This dissertation tries to apply human error research to improve the quality of software artifacts. In order to have the greatest impact on software quality, this research focusses on the very first phase of software development - the requirements phase - wherein customer needs are gathered from different stakeholders and translated into a formal specification. This formal specification is referred to as the SRS (Software Requirements Specification). The SRS is generally written in Natural Language (NL) and acts as a means of communication among stakeholders. The requirements development process (which produces the SRS) is especially prone to human errors (and consequently defects) due to the following reasons: (1) The requirements phase is very fuzzy due to the involvement of a number of technical and non-technical stakeholders like end-users, analysts, and programmers, and (2) Natural language is inherently vague and ambiguous.

As mentioned earlier (in Section 1.1), fault-based defect detection techniques are not able to find all the defects in the requirements artifacts [23], [29]. The major gap left by these fault-based techniques is a missing mapping between a fault (manifestation of a human error) and the underlying source of the fault (i.e., the human error). Therefore, the current dissertation investigates where and how the failures of human cognition (i.e., human errors) occur during the requirements development process. To that end, the primary goal of this dissertation is defined as follows:

> *To identify and analyze the types of human errors that occur during the requirements phase and to develop a structured human error taxonomy to help the requirements engineers in understanding the identified human errors.*

Another goal of this dissertation is to develop techniques and tools that utilize human error information for the purpose of defect detection in requirements artifacts (SRS). With regards to defect detection, the primary focus of this dissertation is on developing a *requirements inspection technique and its supplemental tools*. The two primary goals of this dissertation are reiterated in Table 1.

Table 1. Primary Goals of Dissertation

#	**Goal**
1	Identify and analyze the types of human errors that occur during the requirements phase and develop an intuitive human error framework to help the requirements engineers in understanding the identified human errors.
2	Develop a technique that utilizes the newly developed human error framework for the purpose of defect detection in requirements artifacts (i.e., Software Requirements Specifications).

The rest of this chapter defines the key terms relevant to this research, and describes the research framework developed for achieving the primary goals of this research.

1.3. Definitions

The discussion about software quality revolves around the use of a few important terms: *error* (human error), *fault*, and *failure*. These terms often have been used interchangeably in the software engineering literature. To alleviate confusion, this section provides a definition of each of these terms.

1.3.1. Error, Fault, and Failure

The definitions provided below are consistent with the definitions provided in the IEEE Standard Glossary [30–32]:

> ***Error*** *– A flaw in human thought process that produces an incorrect result, such as software containing a fault. The flaw in human thought process may occur while trying to understand a given information, while solving a problem, or while using a method or a tool. An example of an error in the context of software requirements is: a lack of knowledge about the needs of the user or customer.*
>
> ***Fault*** *– A manifestation of an error within the software. In the context of software requirements, a requirements fault is a manifestation of an error that was committed during the requirements phase of software development.*
>
> ***Failure*** *- Termination of the ability of a product to perform a required function or its inability to perform within previously specified limits. From the perspective of software requirement specifications, failure is defined as the departure of the operational software system's behavior from user expected requirements.*

1.3.2. Software Inspections

Software inspections are one of the most widely used methodologies for verification of software artifacts (e.g., requirements documents) for correctness, consistency, and completeness [13], [33]. The primary goal of software inspections is to uncover faults that get injected during the development of a software artifact. Examples of requirements faults include: incorrect requirement (i.e., a requirement which specifies a user need incorrectly), ambiguous requirement (a requirement that can be interpreted in multiple ways), and inconsistent requirements (i.e., requirements that contradict each other). Software inspections usually consist of three major steps: detection of faults, collection of faults, and repairing the artifact based on the collected faults [21]. A typical inspection consists of the following:

- having inspectors independently review the software artifact to identify faults in the artifact (because this is the most important step in the inspection process, the inspectors need to be trained on a specific inspection technique before this step)
- conducting a team meeting to agree on the faults and compile them in a single list
- sending the list to the author/s of the artifact so that the artifact can be repaired (i.e., the faults identified during the inspection can be removed from the artifact)

1.4. Research Framework

In order to aid seamless integration of *Cognitive Psychology research on human errors* into *Software Engineering research* for attaining improved software quality, this dissertation has three major focal points (also depicted in Figure 1):

I. Creation of a human error taxonomy:

Create the taxonomy using the following major steps:

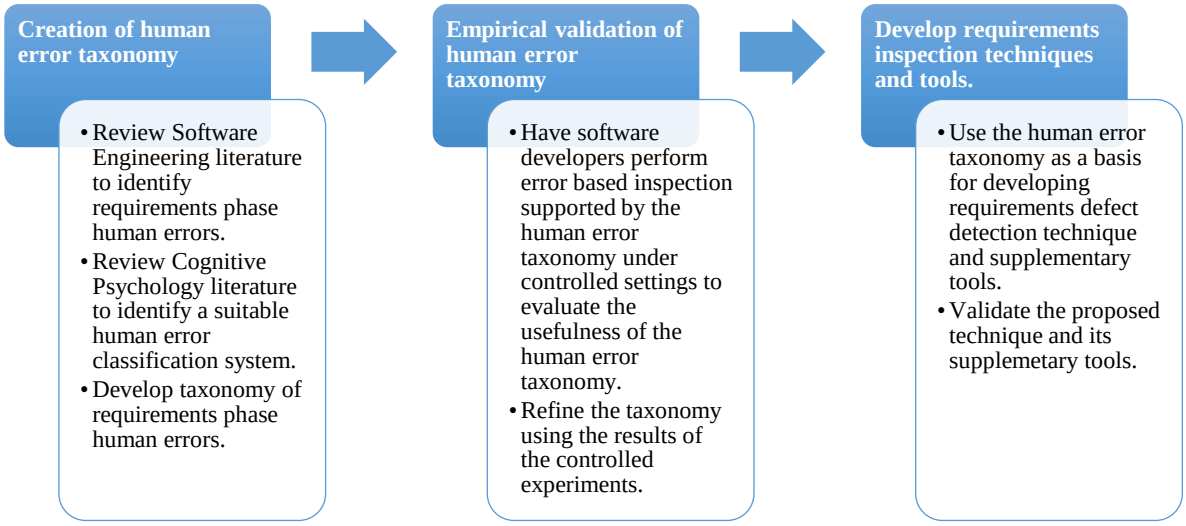


Figure 1. Research Framework

(1) Identify, from the Software Engineering literature, the human errors that occur during the requirements phase of the software development process.

(2) Identify, from Cognitive Psychology literature, a human error classification system that is suitable for creating a taxonomy of requirements phase human errors.

(3) Integrate the human error information collected from Software Engineering literature with the human error theory identified from the Cognitive Psychology literature to create a taxonomy of requirements phase human errors.

II. Empirical validation of the human error taxonomy:

Validate the human error taxonomy as an effective quality improvement approach in controlled experimental settings (academic settings) and refine the human error taxonomy based on the results.

III. Using the human error taxonomy to develop and validate techniques and tools for detecting human errors and faults in software requirements artifacts:

Develop a *requirements defect detection* (i.e., inspection) technique and its supplementary instrumentation. The supplementary instrumentation will include human error tools and trainings to assist inspectors. This part of the research framework will also include empirical validation of the newly developed defect detection approach to evaluate its usefulness during requirements inspections.

2. BACKGROUND RESEARCH

This section describes limitations of the existing quality improvement approaches and how my dissertation aims to overcome those limitations (Section 2.1). Section 2.2 describes the cognitive psychology perspective on human errors and its usefulness in improving software quality.

2.1. Software Quality Improvement Methods

The idea of a requirements inspection approach based on sources of faults (i.e. *errors*) was first proposed by Lanubile et al in 1998 [34]. Lanubile et al conceptualized the idea of *abstraction of errors* from an initial set of requirements faults (found using a traditional inspection approach like the fault-checklist approach), followed by re-inspection of requirements document guided by error information.

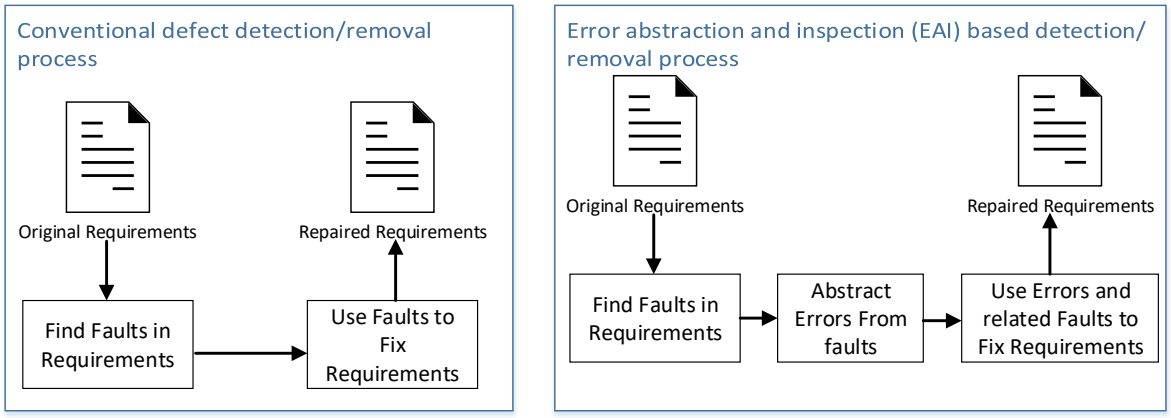


Figure 2. Fault based inspections vs Error Abstraction based inspections

Figure 2 shows the difference between the conventional *fault-checklist based inspection* vs the *Error Abstraction and Inspection* (EAI) approach proposed by Lanubile et al. The EAI requirements inspection approach adds an additional step to the conventional fault checklist based inspections. During the conventional fault based inspection process, first, the reviewers (inspectors) identify faults in the requirements document, and the document author then uses the

reported faults to fix the requirements document. The Error Abstraction and Inspection (EAI) process, however introduces a new step wherein reviewers (inspectors) are asked to abstract errors from the previously discovered faults. The abstracted error information is used to identify additional faults related to the abstracted errors (this step is referred to as error-informed re-inspection). The document author then receives the recovered list of errors and related faults, which he/she then use for repairing the requirements document. Abstracting error information has multiple goals, including but not limited to the following:

- Help inspectors in focusing on those areas of the requirements document that may also have been impacted by errors abstracted during first round of inspection.
- Help in providing an understanding of the real problems that occurred during the requirements development process. These problems are generally left undetected when the focus is on faults alone (traditional inspection approaches only focus on faults and not the underlying errors).
- Abstracted error information helps in providing a better medium for conveying information required for repairing the document. This is because looking for the underlying problems (i.e., human errors) that caused fault-injection allows the document's author/s to learn the actual problems with their requirements development *process* and then prevent the injection of the faults in future.

Lanubile et al also provided evidence of the usefulness of the EAI approach via an empirical study [34]. During the study, a major drawback observed by Lanubile et al in their approach was that the process of error abstraction was heavily reliant on the creativity of reviewers (inspectors) while they are trying to retrospectively trace back the faults to the errors responsible for the faults. In other words, the error abstraction process used in Lanubile's

approach did not provide support for the reviewers (inspectors) during the error abstraction activity. Walia & Carver tried to bridge this gap by proposing an error classification taxonomy that contained information about cause-effect relationship between errors and faults [23].

The error classification taxonomy, titled Requirement Error Taxonomy (RET) was developed for the purpose of guiding the reviewers (inspectors) while they are trying to abstract errors from faults. Another purpose of the RET (Figure 3) was to provide a comprehensive list of the types of errors that generally occur during the requirements development process, so that inspectors do not rely simply on their ability or creativity to think of the errors that are responsible for the faults being analyzed. In order to develop the comprehensive list of requirement errors, Walia & Carver conducted an extensive literature survey [23] that encompassed not only Software Engineering Literature, but also surveyed human cognition research to evaluate if any of the human errors proposed in human cognition research can have corresponding errors in software requirements. The result of the literature survey conducted by Walia & Carver was the Requirement Error Taxonomy (shown in Figure 3), which grouped

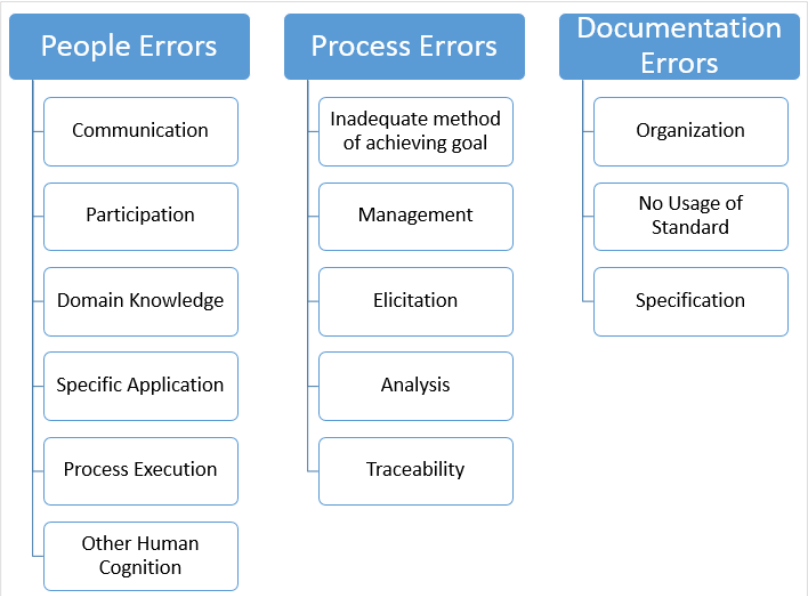


Figure 3. Requirement Error Taxonomy (RET)

requirement errors into three major categories: *People Errors*, *Process Errors*, and *Documentation Errors*. The *People Errors* category describes errors caused by the fallibilities of the people involved in the requirements development process. The *Process Error* category describes errors caused by selection of incorrect or inappropriate requirements engineering technique or process. *Documentation Error* category describes errors caused due to incorrectly organizing and specifying the requirements.

Following the development of the RET, Walia & Carver conducted multiple empirical studies [29, 35–37] with the goal of: (1) validating whether the Error Abstraction and Inspection (EAI) process is an effective approach for identifying faults in requirements documents, (2) validating whether the Requirement Error Taxonomy or RET is a useful addition to Lanubile's EAI process, and (3) validating if integration of cognitive psychology research with software engineering research is useful for software quality improvement. The results from the empirical studies conducted by Walia & Carver [29] highlighted the following key observations:

- The results of the empirical studies validated that not only does EAI (guided by RET) provides an improvement in inspectors' fault detection effectiveness when compared to fault-based inspection, RET was also found to have a very positive impression on the participants of the studies with regards to its usefulness in helping them locate faults.
- The results of the empirical studies highlighted that the *People Error* class (errors caused by fallibilities of individuals involved in requirements development) of RET was reported to be the source of a significantly large number of faults (up to 32% of the faults were traced back to *People Errors*).

The observation that *People Errors* were found responsible for a majority of faults in requirements documents was of particular interest and a major motivational factor for the research described in the current dissertation. Although, RET was found to be useful and effective, RET was lacking in a direct tie-in with the research performed by Cognitive Psychologists on human errors and the psychological processes that produce human errors. Furthermore, even though RET did try to identify human errors that could have corresponding requirements errors, it failed to evaluate each of the identified errors from the perspective of whether or not the error qualified as a failure of human cognition (i.e., human error).

A major limitation of RET was that it was created without reference to psychological theories of how human errors occur, and the observation that *People Errors* of RET were found to be responsible for a significantly large number of faults were the two primary motivational factors behind conducting the current research. The current research proposes to extend the work done by Walia & Carver on RET by creating a *requirements phase human error taxonomy* which is more strongly grounded in *human error theories* proposed in the Cognitive Psychology literature. Section 2.1 provides a brief review of the formal literature on human errors from a psychological perspective and the relevance of human error theories to Software Engineering domain.

2.2. A Cognitive Psychology Perspective on Errors

As defined in Section 1.3, the term *error*, in context of the current research is understood as failings of human cognition in the process of perception, judgement, problem-solving, decision-making, planning, and execution of the plan (i.e., acting). This failure of human cognition is often referred to as *human errors* or *mental errors* in the Cognitive Psychology literature. These errors, in turn produce faults, which are physical manifestation of error. These

faults, when left undetected, may result in *system failure*. This is to say that most of the system failures find their origin in a *human cognition failure* or *human error*.

Human errors have been examined in domains such as aviation, medicine, and the oil industry. Cognitive Psychologists have developed domain-specific taxonomies of human errors that capture systematic failure in human-performance [2, 4, 24, 26–28, 38]. The commonality between the domain-specific taxonomies is that these taxonomies capitalize upon elementary theoretical research on human cognition. Often, these elementary human cognition theories or human error theories employ a human information-processing model that provides a coherent account of human errors that are committed by people when they are performing different tasks. Cognitive Psychologists argue that human errors generally are not the result of irrational or maladaptive tendencies, but instead result from "*normal, useful psychological processes gone awry"* [39]. Hence, human errors can be organized around the normal psychological processes that an individual goes through when performing any task.

Similar to domains like aviation, medicine etc., Software Engineering is a process dependent upon human operators and hence is susceptible to human errors (human cognition failures). Human errors in software engineering arise during information processing, particularly as information is translated from one form to another (for example, during requirements development when customer needs are translated into formal requirements specifications). The current research proposes that application of human error research to Software Engineering offers great promise for reducing faults and improving software quality.

Cognitive scientists have proposed several human error classification systems e.g., Human Factors Analysis & Classification system (or HFACS) [27], 'Swiss Cheese' model [39], Reason's and Rasmussen's taxonomy [39–43], and Norman's classification [44] to show that

people's judgments and decision-making can be erroneous when faced with different situations. The most prominent and widely respected human error classification system is the one proposed by James Reason [3, 4, 26–28]. James Reason, in his seminal book titled, Human Error [39], proposed a general classification system of human errors, wherein human errors are organized around a very simple model of human information-processing. Reason proposes that there are three general processes a human operator goes through in order to perform an action: (1) sensing and perceiving information, (2) processing the information and making a decision about which action/s to take, and (3) taking action. Figure 4 shows the general human information-processing model proposed by Reason. Human errors can originate in any of the three processes shown in Figure 4.

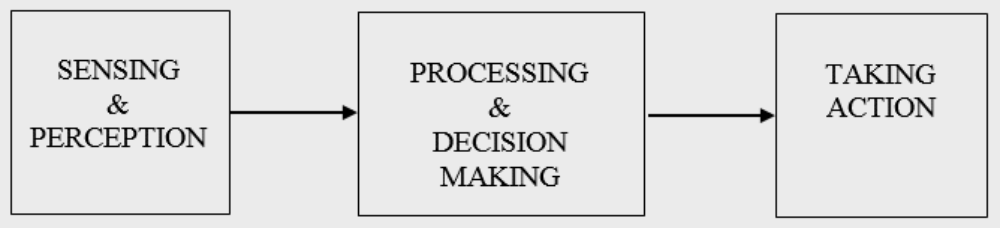


Figure 4. General Human Information-Processing Model Proposed by Reason

Reason defines the human errors associated with the three processes into three broad categories: *slips*, *lapses*, and *mistakes*. *Slips* and *lapses* happen while executing the planned action (taking action), whereas *mistakes* happen during the perception, processing, and decision making stages. Reason further provides the underlying cognitive failure mechanisms behind slips, lapses and mistakes (shown in Figure 5). As can be seen in Figure 5, slips and lapses occur due to inattention and memory failures, respectively, and mistakes occur due to inadequate formulation of plan. Inadequate formulation of plan often is due to lack of knowledge or inadequate application of procedures/rules to a given situation.

Reason's classification establishes a very strong association between the human information-processing model and cognitive failure mechanisms like inattention, memory failures, and lack of knowledge. This makes Reason's human error classification system easily applicable and adaptable to any domain. Another benefit of applying Reason's model of slips, lapses and mistakes to the Software Engineering domain is that the information-processing stages software engineers undergo are similar to the ones on which Reason's classification is built upon: process user needs, make decisions about project plan, and execute (implement) the plan.

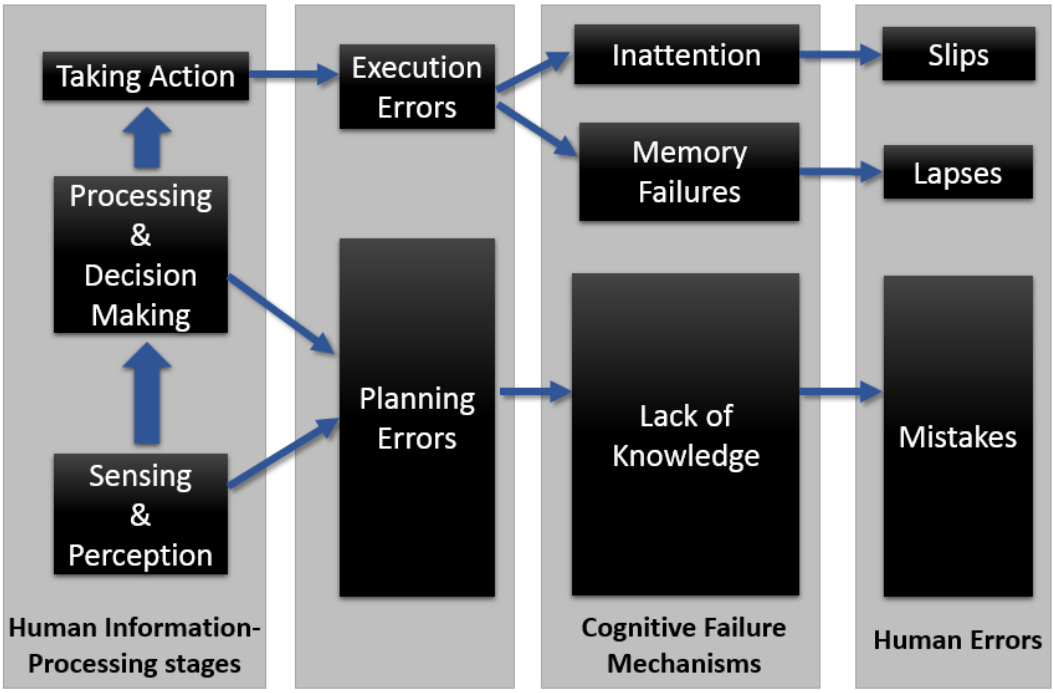


Figure 5. Human Error Model Proposed by Reason

The current research makes an effort to adapt the theoretical research on human errors (specifically the research done by James Reason) to the software engineering domain. The current research also provides evidence that such an adaptation will provide an improved understanding of the means to detect and remove software engineering human errors and the corresponding faults, and consequently have a positive impact on software process & product

quality. The next chapter provides the development process of a requirements engineering human error taxonomy in which requirements engineering human errors are organized around Reason's slips, lapses, and mistakes.

3. IDENTIFICATION AND CLASSIFICATION OF REQUIREMENTS ENGINEERING HUMAN ERRORS

This chapter describes the research approach used to develop the Human Error Taxonomy (HET) that describes the most commonly occurring requirements engineering human errors and classifies them into Reason's slips, lapses, and mistakes. Section 3.1 describes the systematic literature review as a research method for developing the HET, and Section 3.2 provides the major outcomes of the systematic review process.

3.1. Systematic Review Process for Developing Human Error Taxonomy

This systematic literature review was conducted to identify and classify the requirements phase human errors reported in the Software Engineering (SE) literature. Systematic Literature Review (SLR) is a specific methodology of research, which was developed in order to gather and analyze the available state of knowledge pertaining to a topic [45–47]. As the name suggests, a systematic literature review follows a very well formulated and strictly structured method, referred to as the *review protocol*. The review protocol is created by:

- first, expressing the focused research topic (which is being investigated) as one or more structured question/s using specific terms and concepts that are relevant and must be addressed in order to collect as much information about the topic as possible, and
- second, creating strategies to retrieve the information around the pre-defined structured question/s.

The review protocol needs to be explicitly defined so that other researchers can reproduce the same procedure and be able to evaluate whether the protocol defined for the focused topic is adequate to retrieve as much information about the topic as possible.

An effective systematic review is one that is driven by an overall goal. In the current systematic review, the high-level goal was:

What types of human errors that occur during the requirements phase can be identified in the literature and how can human error theories help in creating a classification system for those human errors?

The high-level question was further decomposed into two detailed research questions. The purpose of the first detailed research question was to identify the human errors reported in Software Engineering (SE) literature, and the purpose of the second research question was to organize the identified human errors into a taxonomy. Table 2 provides the two detailed research questions.

Table 2. Research Questions for the Systematic Literature Review

#	**Research Question**
RQ1	What types of requirements engineering human errors does the software engineering and psychology literature describe?
RQ2	How can we organize the human errors identified in RQ1 into a taxonomy?

Subsections 3.1.1, 3.1.2, and 3.1.3 describe the rest of the review protocol (Source Selection, Primary Study Selection, and Data Extraction).

3.1.1. Source Selection and Search

The systematic review that I conducted was an extension and replication of the review conducted by Walia et al [23]. Therefore, to identify relevant publications, I used the same search strings that were used by Walia et al. The search strings were executed in IEEExplore, INSPEC, ACM Digital Library, SCIRUS (Elsevier), Google Scholar, PsychINFO (EBSCO), and Science Citation Index. Because Walia et al included papers published through 2006 in their

review, I only searched for studies published after 2006 (through October 2014). Table 3 provides the detailed search strings.

Table 3. Search Strings [23]

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))

3.1.2. Study Selection

During the first phase, called title-elimination phase, a total of 280 studies were identified by running the search strings on the various publication databases. Next, the abstracts and keywords were read to exclude any studies that were clearly unrelated to the research questions. At the end of this stage, the list contained 96 studies.

Next, the full-text of the 96 remaining studies was reviewed. The inclusion/exclusion criteria shown in Table 4 was used to examine the full-text of the 96 studies. This step resulted in 34 studies remaining.

Table 4. Inclusion-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 the 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 the psychology literature about models of human error. - Papers from the cognitive psychology literature about human the thought process, planning, human reasoning or problem solving. - Empirical studies (qualitative or quantitative) on human errors. - Papers describing various human error classification systems.	

Next, four additional studies were identified through snowballing (i.e., searching through the references of the 34 remaining studies). This step resulted in 38 included studies. Fig. 6 summarizes the search and selection process.

3.1.3. Data Extraction

Based on the type of information contained in the primary study, we classified 11 primary studies as Cognitive Psychology studies and 27 studies as Software Engineering studies. The Cognitive Psychology papers focused on models of human errors. The Software Engineering studies primarily focused on human errors committed during the process of software

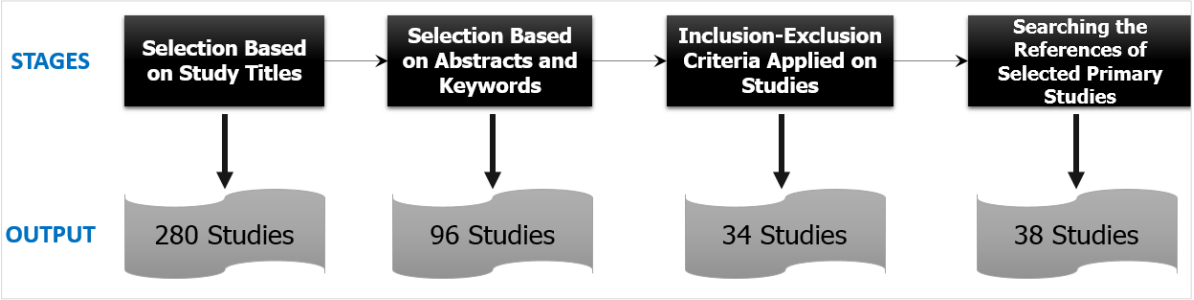


Figure 6. Study Selection Process

development. The content and focus of these two types of studies was different, and hence different types of data was extracted from them.

Note that studies included by Walia et al in their review [23] were reanalyzed and data was extracted from these studies as well. Table 5 lists the common data extracted from all studies. Table 6 lists the data extracted based on each primary study's research focus. Using these data extraction forms, information was extracted from the studies.

Table 5. 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	Relates to one/more search focus, i.e., research area(s) the paper focuses 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

Table 6. Data Items Related 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 requirement 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?
	Requirement phase	Yes or No (whether it was applied in requirement 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 requirement 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 requirement errors

3.2. Results of the Systematic Review Process

This section is organized around the two research questions (shown in Table 2) that were driving this systematic literature review.

3.2.1. RQ1: Type of Requirements Engineering Human Errors Described in Literature

To start with, individual errors, error categories, error-descriptions, and root causes were extracted from the 38 primary studies. Similarly, the published systematic review paper written by Walia et al [23] was analyzed and errors and their descriptions were extracted. An examination of this extracted information revealed multiple interpretations of the term 'error'. These interpretations included: software defects, program errors, requirements defects, requirements problems, end-user (or user) errors, and human error.

Before building the taxonomy, items that were not truly human errors were removed. The output of this process was 31 human errors, listed in Table 7.

At this point, the list of included studies in this review was also updated. First, only seven out of the 27 software engineering primary studies contained a true requirements engineering human error. Therefore, the other 20 studies were eliminated. Second, none of the Psychology studies contained any requirements engineering human errors. Hence, these studies were also excluded.

Additionally, there were eleven (11) studies from the review paper published by Walia et al [23] that identified requirements engineering human errors. So, in total, eighteen (18) studies from the software engineering literature were included as they contain true requirements engineering human errors (see Appendix A for a list of the 18 studies that provided input to the Human Error Taxonomy).

Table 7. Human Errors Identified in Literature

Error #	Error Name	Source (see Appendix A)
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 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 2006
10	Mistaken belief that it is impossible to specify NFRs 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	Walia et al 2009
15	Environment errors	Walia et al 2009
16	Information Management errors	Walia et al 2009
17	Lack of awareness of sources of requirements	Walia et al 2009
18	Application errors	Walia et al 2009
19	Requirements developer did not understand some aspect of the product or process	Walia et al 2009
20	User needs not well-understood or interpreted by different stakeholders	Walia et al 2009
21	Lack of understanding of the system	Walia et al 2009
22	Lack of system knowledge	Walia et al 2009
23	Not understanding some parts of the problem domain	Walia et al 2009
24	Misunderstandings caused by working simultaneously with several different software systems and domains	Walia et al 2009
25	Misunderstanding of some aspect of the overall functionality of the system	Walia et al 2009
26	Problem-Solution errors	Walia et al 2009
27	Misunderstanding of problem solution processes	Walia et al 2009

Table 7. Human Errors Identified in Literature (continued)

Error #	Error Name	Source (see Appendix A)
28	Semantic errors	Walia et al 2009
29	Syntax errors	Walia et al 2009
30	Clerical errors	Walia et al 2009
31	Carelessness while documenting requirements	Walia et al 2009

Table 8. Human Error Taxonomy (HET)

Reason's Taxonomy	Human Error Class	Human Error(s) from Table 7
Slips	Clerical Errors	30, 31
	Lack of consistency in the requirement specification	5
Lapses	Loss of information from stakeholders	6
	Accidentally overlooking requirements	11
Mistakes	Application Errors	18, 25
	Solution Choice Errors	26, 27
	Syntactic Errors	28, 29
	Wrong Assumptions	3, 4, 14
	Environment Errors	15
	Information management Errors	16
	Poor understanding of one another's roles	8
	Not having a clear distinction between client an users	9
	Mistaken belief that it is impossible to specify non-functional requirements in a verifiable form	10
	Inadequate requirements process	13
	Lack of awareness of requirements sources	17
Violations	Assumption that insufficient requirements are ok	7
	Ignoring some requirements engineering tasks	12

3.2.2. RQ2: Organizing the Human Errors Identified in RQ1 into a Taxonomy

To create the human error taxonomy (HET), two steps were followed. First, the individual errors in Table 7 were examined and grouped into classes based on similarities.

These analyses were performed by studying the description provided for the error by the primary study that supplied the error. Human error expert, Dr. Gary Bradshaw (Professor, Cognitive Science Program, Mississippi State University) evaluated the final classification, which is shown in Table 8.

Then, the error classes were organized into Reason's Slips, Lapses, Mistakes taxonomy. For this organization, each error class was analyzed to (i) decide whether it was a planning or an execution error and (ii) for execution errors, decide whether the error was related to attention failures (slips) or memory failures (lapses).

Note that the last two rows of Table 8 contain intentional violations (i.e., deliberately failing to follow rules), which are different from unintentional errors (slips, lapses, and mistakes). The human error taxonomy covers only unintentional errors. Hence, the taxonomy excludes violations. Figure 7 shows the final outcome (i.e., the Human Error Taxonomy) of answering RQ1.

In order to make the error classes in Table 8 (or Figure 7) more understandable, a description of each class along with an example error and fault is provided in Tables 9, 10, and 11. For the examples given in the tables, the Loan Arranger (LA) system is used. The

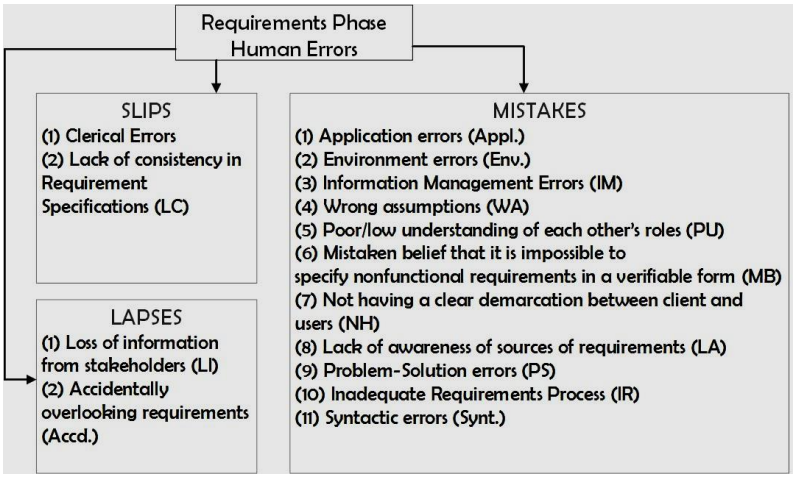


Figure 7. Human Error taxonomy (HET)

requirements for Loan Arranger (LA) system were developed by researchers for use in software quality research. A brief overview of LA system is provided in the next paragraph.

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 system automates information management activities, such as updating loan information monthly.

Table 9. Slip Errors in Human Error Taxonomy (HET)

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: The requirement author understood the difference between regular loans (amount <= $275,000) and jumbo loans (amount > $275,000). But, while documenting the requirements, s/he recorded the same information for both types of loans.	Fault: The requirements for the jumbo loans incorrectly specify the same behavior as for regular loans
Lack of consistency in the requirement specification errors	Occur when requirement authors do not articulate or organize the requirements in a consistent manner, even when they have a clear idea of user needs. This error leads to a disjointed requirements specification, which makes interpretation difficult.	Error: The requirement author is not consistent with his/her use of terminology. 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.

Table 10. Lapse Errors in Human Error Taxonomy (HET)

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 reports (file, screen, or printout) from a loan analyst, but 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: Requirement document does not describe the process of system recovery from abnormal termination.

Table 11. Mistake Errors in Human Error Taxonomy (HET)

Error Name	Description	Example of error	Example of fault
Application errors	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 told her all information required 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	Result from lack of knowledge about the available infrastructure that supports the development of a given project. This infrastructure includes tools, templates, or other items of infrastructure that support the elicitation, understanding, or documentation of software requirements.	Error: The requirement authors did not use a standard template for documenting the requirements (for example, the IEEE standard template for SRS) because they were unaware of the presence of such a template. Therefore, the author did not use right tools.	Fault: Requirements about system scope and performance were omitted.
Information Management errors	Result from a lack of knowledge about standard requirement engineering or documentation practices and procedures within the organization	Error: It is common procedure within the organization that requirement specifications include error-handling information about which mechanisms are invoked when error occur. This specification does not contain any information about 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 notification.
Wrong Assumption errors	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.
Poor understanding of one another's roles	Domain knowledge and perspectives vary between roles, which necessitates considerable communication among members of the software engineering team. Without proper understanding of developer roles, communication gaps may arise, either by failing to communicate at all (due to lack of understanding that other roles are impacted) or by ineffective communication (e.g. missing tacit requirements due to lack of insight into the customer's domain.)	Error: It was not clear among team members, who needed to elicit the requirements of 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.

Table 11. Mistake Errors in Human Error Taxonomy (HET) (continued)

Error Name	Description	Example of error	Example of fault
Mistaken belief that it is impossible to specify non-functional requirements in a verifiable form	Major causes of this problem are the prevalent myths that it is too costly, too difficult, and even impossible to produce good requirements, especially nonfunctional requirements, during the software engineering process. These myths are especially prevalent with regard to quality and specialty engineering requirements (e.g., availability, interoperability, performance, portability, safety, security, and usability), where there is still a prevailing but mistaken belief that it is impossible to specify these requirements in a verifiable form.	Error: An absence of any performance, security, usability, or availability requirements suggests that all non-functional requirements were overlooked.	Fault: Omission of performance requirements, security requirements and other non-functional requirements.
Not having a clear distinction between client and users	If RE practitioners are not able to distinguish between clients and end users, or do not realize that the clients are distinct from the 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 whereas 'Purpose' specifies loans can be edited.
Lack of awareness of requirement sources	Requirements gathering person is not aware of all stakeholders which he/she should contact in order to gather the complete set of user needs. Sources of requirements include all different types of end users of the system being built and all the decision-makers from project sponsoring organization (also called the customers or the clients)	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.
Solution Choice (or Problem Solution) errors	Are due to not knowing, misunderstanding, or misuse of problem solution processes. This kind of 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 methods and techniques, they might end up analyzing the problem incorrectly, and choose the wrong solution	Error: Lack of knowledge of the requirement engineering process and requirement engineering terminology on part of the analysts. The analyst does not understand what kind of requirements are performance requirements and what kind of requirements are functional requirements.	Fault: A particular requirement listed under performance requirement should be a functional requirement.
Inadequate Requirements Process	Errors occur when the requirement authors do not fully understand all of the requirement engineering steps necessary to ensure the software is complete and inadvertently omit one or more steps from the plan.	Error: Requirement engineering plan did not have sufficient 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.
Syntax errors	Occur when a requirement author misunderstand 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.

3.3. Evaluating the Usefulness of Human Error Taxonomy (HET)

After the development of the HET, its usefulness for error and fault detection was evaluated via empirical studies that were conducted in academic settings. From a requirements

defect detection perspective, my research proposes that, a deeper understanding of the errors that affected the development of a particular requirements artifact can lead requirements engineers to detect faults that are often overlooked during traditional inspections. To that end, feasibility studies were conducted to determine whether software developers can use the HET to improve their defect detection ability during a requirements inspection. The next chapter describes the design and execution of the studies that evaluated the usefulness of HET for requirements defect detection.

4. VALIDATION OF THE HUMAN ERROR TAXONOMY

As mentioned earlier, the Human Error Taxonomy (HET) was constructed with the goal of using the taxonomy as a basis for developing requirements defect/fault detection (i.e., requirements inspection) techniques. Therefore, a primary goal of my dissertation is to develop and empirically validate human error based (i.e. HET based) requirements inspection (i.e., fault detection) tools and techniques. In order to evaluate the usefulness of HET as a requirements fault detection tool, a formal Error Abstraction and inspection (EAI) approach is employed. The EAI approach adds an additional step to the traditional Fault Checklist (FC) based inspection. The extra step consists of assisting inspectors in identifying underlying human errors (abstracted from the faults found during the FC inspection). Inspectors then use the abstracted human error information to re-inspect SRS (Software Requirements Specification document) for additional faults.

The usefulness of the human error taxonomy (HET) during requirements inspections was evaluated in three controlled experiments, two of which were conducted at North Dakota State University (NDSU), and one was conducted at University of Alabama (UA) at Tuscaloosa. While the major goal of the studies was same - to evaluate the usefulness of human errors taxonomy for requirements fault detection – the designs were slightly different from each other to gather insights about different aspects of using human error taxonomy (HET) for requirements fault detection (example of these aspects include relevance of HET's human error classes to requirements engineering).

This chapter first discusses, in Section 4.1, the overarching Research Questions that these three controlled studies answered. Next, in Section 4.2, details about each study's design are

provided. Section 4.3 provides the results of analyzing the data gathered during the three studies. Essentially, Section 4.3 provides the answers for the research questions described in Section 4.1.

4.1. Research Questions for Empirical Validation of the Human Error Taxonomy

As mentioned earlier, this section provides the main research questions that were formulated to enable data collection for evaluating the usefulness of the human error taxonomy for requirements fault detection. Table 12 provides the Research Questions that the three empirical studies (described in this chapter) provided data for.

Table 12. Research Questions for Evaluating Usefulness of the HET

#	Research Question
RQ1	Does the Human Error Taxonomy improve the fault detection effectiveness of inspectors when compared to existing requirements inspection techniques?
RQ2	Does the Human Error Taxonomy provide a useful method of understanding and classifying the human errors and faults made during development of a Software Requirements Specification document?

4.2. Description of Designs of the Three Empirical Studies

This section provides the designs of the three controlled studies that provided data for answering the research questions described in Table 12. In this section, designs of the three studies are provided, and next in Section 4.3, the data analysis and results (based on the research questions shown in Table 12) are provided.

4.2.1. Experiment Design for Study 1 (A Control Group Study)

Study 1 compared the fault (or defect) detection effectiveness of the Human Error Taxonomy with that of an existing error taxonomy called Requirement Error Taxonomy. The Requirement Error Taxonomy (RET) [23] was an initial foray into error taxonomy based inspections, but it was found that RET lacked a strong grounding in Cognitive Psychology

theories. This lack of connection with Cognitive Psychology theories was one of the motivations behind creation of the Human Error Taxonomy. Study 1 was designed to evaluate if the updated human error taxonomy (i.e., HET) offers an improvement over RET (a proven verification technique) during requirements inspection. To that end, a randomized pre-test post-test control group experiment was executed in controlled settings. The control group used RET, whereas the experimental group used the newly developed HET to perform requirements inspection.

The participants in this study were 46 computer science students, enrolled in the *Principles of Software Engineering* course at North Dakota State University (NDSU). The course required students to work in teams (teams were selected by the instructor prior to this study) to develop Software Requirements Specification (SRS) documents for different software systems. To enable a comparison between HET vs. RET, participants were randomly divided in each team into two equal groups (a control group that used RET and an experiment group that used HET). Figure 8 shows the division of participants into three teams (e.g., team 1 had 16 participants) and subdivision of each team into treatment groups (8 used RET and 8 used HET).

During the training (*pre-test*), the participants (23 in experimental group and 23 in control group) were trained on their respective taxonomies (HET for experimental and RET for control group) by having them perform an error based inspection of an external SRS document that was seeded with 30 realistic faults. The SRS used during the training specified requirements for a Parking Garage Control System (PGCS). During the *post-test*, participants inspected the SRS's that they had developed (as part of a team). Members of Team 1 developed and inspected the SRS for *Fly-by* system, an airline reservation and travel management system. Team 2 developed and inspected the SRS for *Campus Reconnection* system, a student information and course management system. Team 3 developed and inspected the SRS for FaceSpace system, an online

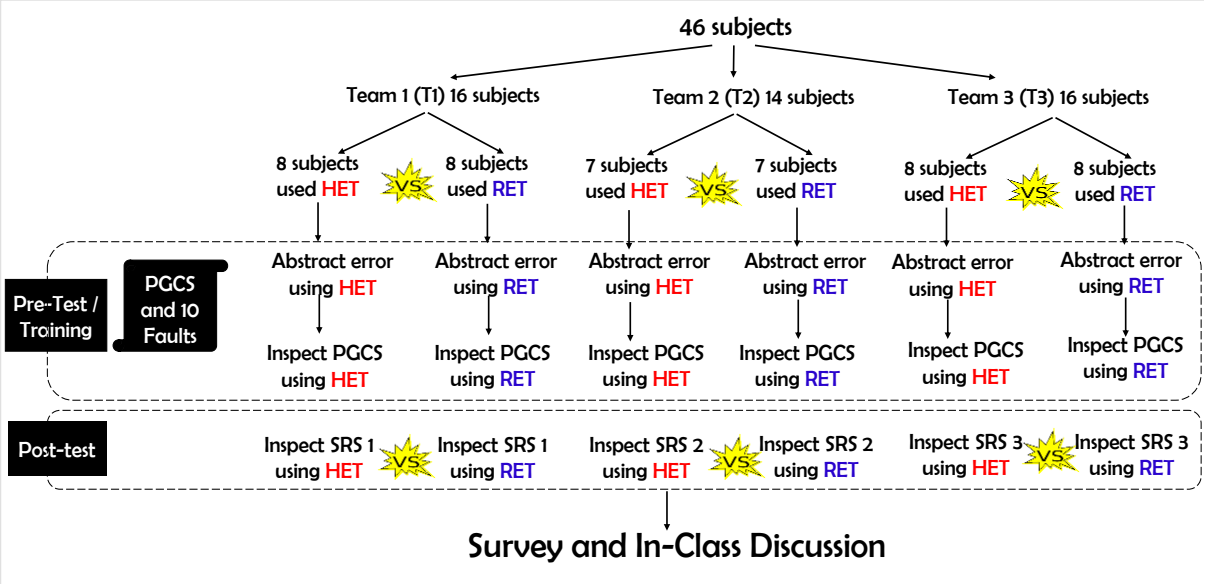


Figure 8. Experiment Procedure: Assignment of Participants, Artifacts and Output

music streaming system. As shown in Figure 8, half of the participants within each team inspected their own SRS using HET (e.g., 8 in team 1) or RET (other 8 in team 1), depending on the treatment group they were assigned during the *pre-test*. A detailed descriptions of the various experimental steps performed by participants appears in Table 13.

Table 13. Steps Performed by Participants during Study 1

Experimental Step		**Description**
Pretest Steps	Training on HET and RET	Participants were trained on the requirements inspections and the different type of requirement faults. Next, in two separate sessions, 23 participants were trained on HET and 23 participants were trained on RET. The training involved teaching the participants about the error abstraction using HET/RET (i.e., how to identify errors from faults), and using the abstracted error information to perform fault inspection (i.e., how to locate new faults).
	Error Abstraction	After the training, participants were given 10 faults in PGCS SRS (chosen randomly from 30 seeded faults). Participants then used HET/RET to abstract and classify errors from the 10 given faults. This step resulted in 46 error forms (23 for RET and 23 for HET).
	Fault Inspection	The participants then used the abstracted error information (from error forms) to locate additional faults in the PGCS SRS (i.e. participants re-inspected PGCS SRS using errors). This step resulted in 46 fault-forms (23 for HET and 23 for RET) containing new faults in PGCS SRS.
SRS development		Participants then worked in their respective teams (three teams) to develop requirements specification documents or SRS for different systems.
Post-test steps: error-inspection on self-created SRS		During the post-test, each participants inspected their own SRS (which they had developed as a team) using the technique that were trained during the pre-test (HET or RET) and reported faults. For example, of the 16 participants in Team 1, 8 used HET while the other 8 used RET to inspect the "Fly-by" SRS. This step produced 46 individual fault-forms.
Post-study Survey and Focus Group		The experimental group and the control group participants rated HET and RET across various usefulness categories on a 5-point scale (ranging from "1 – not useful" to "5 – very useful"). A focus group discussion was conducted in order to understand the problems faced by participants while using HET/RET to find faults.

4.2.2. Experiment Design for Study 2 (A Feasibility Study)

Similar to the control group study described in Section 4.2.1, the major goal of Study 2 (which was conducted at University of Alabama at Tuscaloosa) was to evaluate the feasibility of using the Human Error Taxonomy (HET) to support the requirements inspection process. Study 2 also focused on analyzing whether the HET is useful for classifying errors and for guiding inspectors to find additional faults.

Table 14. Steps Performed by Participants during Study 2

Experimental Step	Description
Training 1 - Fault checklist technique	During this 15-minute training, participants were trained on how to use the fault checklist technique to inspect an SRS document.
Step 1- First Inspection (Individual Inspection)	Each participants was randomly assigned an SRS developed by another team. The participants used the fault checklist technique (from Training 1) to inspect (i.e., identify faults) in the assigned SRS. The output of Step 1 was 28 individual Fault Forms (one per participants).
Step 2 - Team Meeting to Consolidate Faults	Each team were provided with the individual Fault Forms submitted by the participants who inspected their SRS (from Step 1). Each team then worked as a group to examine these fault lists to first remove duplicates and then consolidate the faults into one single master list, which they documented on the Group Fault Form. The output of Step 2 was eight Group Fault Forms (one per team).
Training 2 - Error abstraction and classification	During this 40-minute session, participants were first trained on the error abstraction process, and then trained on the HET and how to use HET to abstract and classify requirements errors.
Step 4 - Error-informed Re-inspection of the SRS	Using the errors abstracted during Step 3, each participants individually re-inspected their own SRS to identify any additional faults related to these errors (i.e. the faults that were not found by their classmates during Step 1). The participants documented the additional faults on a Re-inspection Form. The output of Step 4 was 28 individual Re-inspection Forms (one per participants).
Step 5 - Coordinating the individual fault lists	This step is similar to Step 2, except that each team used the faults found during the re-inspection (Step 4). Each team created a Final Group Fault Form, in which they reported the agreed-upon list of faults. The output of Step 5 was eight Final Group Fault Forms (one per team).
Post-study Survey	After completing all experimental steps, each participants provided feedback about the error abstraction process and the HET.

The study's participants were 28 senior-level undergraduate computer science students. The students were enrolled in the Fall'15 capstone project course at University of Alabama at Tuscaloosa. The primary goal of this course was for the student teams to undergo the entire software development process (requirements elicitation/documentation, design, implementation, and testing) in order to build a complete software system. Students were divided into eight three

or four-person teams by the course instructor (not part of the research team). Each team developed their own system.

The inspection artifacts were the 8 SRS's developed by the teams. A detailed description of the various experimental steps performed by participants during Study 2 appears in Table 14.

The complete experimental package used for Study 2 can be found here: http://humanerrorinse.org/Studies/2015/Fall_UA.

4.2.3. Experiment Design for Study 3 (Study to Evaluate the Educational Value of HET)

Thirty-four (34) graduate students enrolled in the Software Development Processes course in North Dakota State University participated in Study 3. Students were trained on the Human Error Taxonomy (HET), and how to abstract human errors from faults. The primary focus of Study 3 was to evaluate whether performing error abstraction on faults found in an externally developed requirements document can help students understand requirement phase human errors. Table 15 provides the experimental steps performed during this study.

Table 15. Steps Performed by Participants during Study 3

Experimental Step	Description
Training - Error abstraction and classification	During this 50-minute session, participants were trained on the HET, and how to use HET to abstract and classify requirements errors from faults.
Error abstraction from faults in PGCS SRS.	The students were given 10 randomly selected faults (from 30 seeded faults) in PGCS SRS and asked to analyze these 10 faults to abstract and classify human errors into one of the error class of HET. The result of this step was 34 individual error lists containing human errors (and their classifications) that may have occurred during creation of PGCS requirements
Post-study survey	The survey gathered students' feedback on HET and their understanding of the human errors and cognitive failure mechanisms that affect the requirements development process

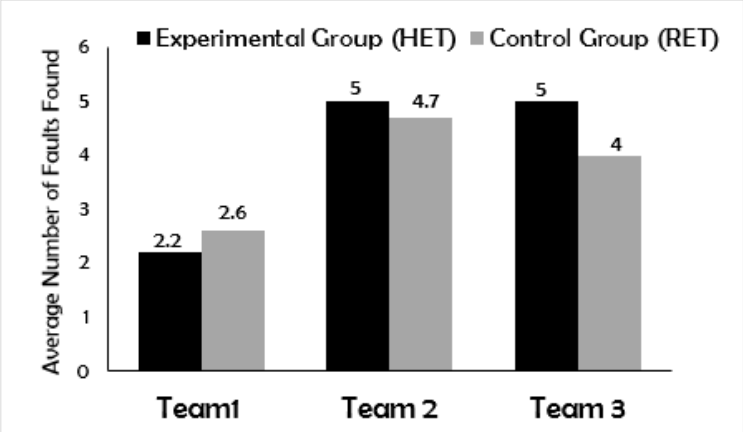


Figure 9. Study 1: Comparison of Average Number of Faults

During the survey, participants rated usefulness of HET on a 5-point scale (ranging from "1- Strongly Disagree" to "5 – Strongly Agree"). The first category of survey questions evaluated students' abilities to distinguish between human error types (slips, lapses, and mistakes). A second category of survey questions evaluated the usefulness of HET in helping students understand the requirement phase errors and the faults caused by the errors.

4.3. Analysis of Data Gathered During Studies 1, 2, and 3

This section provides the results of analyzing the data gathered during Studies 1, 2, and 3. This section is organized around the two research questions (see Section 4.1) that were formulated to validate the usefulness of the Human Error Taxonomy for requirements fault detection. The three studies (Studies 1, 2, and 3) were designed with the goal of collecting the data to answer the research questions described in Table 12. Experimental design for each of the three studies were provided in Sections 4.2.1, 4.2.2, and 4.2.3, respectively.

4.3.1. RQ1: Does the Human Error Taxonomy Improve the Fault Detection Effectiveness of Inspectors when Compared to Existing Requirements Inspection Techniques?

Data gathered during Study 1 and Study 2 was analyzed to answer this research question. Study 1 was a control group study that compared the fault detection effectiveness provided by

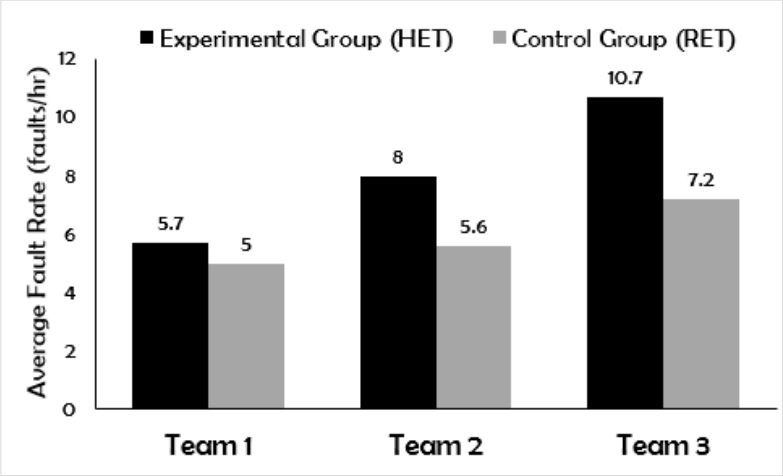


Figure 10. Study 1: Comparison of Average Fault Rate or Efficiency (faults/hour)

HET *vs* fault detection effectiveness provided by RET (an existing requirements verification technique). During Study 1, experimental group participants who used HET found more faults than the control group participants who used RET for two out of three teams (as shown in Figure 9). In terms of efficiency (faults per hour), during Study 1, participants who used HET (experiment group) found faults at a much faster rate than the participants who used RET (control group) for all three teams (as shown in Figure 10).

Independent sample t-tests were run for both effectiveness and efficiency for all three teams. It was found that although the participants who used HET generally performed better than the participants using RET, the effectiveness and efficiency improvement was not statistically significant. With respect to effectiveness, the p-values obtained during the independent measures t-tests were 0.684, 0.866, and 0.705 for Teams 1, 2, and 3 respectively, indicating that HET group did not found significantly more faults. With respect to efficiency, the p-values obtained during the t-tests were 0.835, 0.536, and 0.608 for Teams 1, 2, and 3 respectively indicating that the efficiency for HET group was not significantly better than RET group.

Study 2 was a feasibility study that evaluated whether software development teams are able to use human error information to find additional faults that they were not able to find during a traditional fault-checklist based inspection. Results (which are provided in Figure 11) from analyzing the data gathered during Study 2 showed that, all six teams found additional faults during the re- inspection using abstracted error information, but the number of additional faults found was not higher than the number of faults found during fault-checklist based inspection (this maybe because most of the faults were already found during first inspection). But overall, all teams located new faults that were not located during fault-checklist inspection, thereby improving quality of their requirements.

Results from Study 1 and Study 2 show that the Human Error Taxonomy helped improve the fault detection effectiveness of inspectors when compared to the existing techniques (Requirement Error Taxonomy and Fault-checklist based inspection techniques).

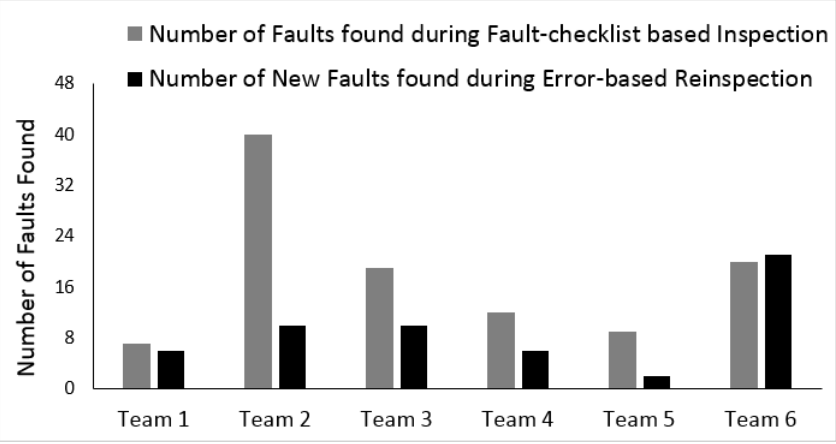


Figure 11. Study 2: Number of New Faults Found During Error-based Reinspection

4.3.2. RQ2: Does the Human Error Taxonomy Provide a Useful Method of Describing and Classifying the Human Errors and Faults Made During Development of a Software Requirements Specification document?

This research question was further broken down into two research questions:

- RQ2a: Are all three Error types (slips, lapses, mistakes) and all error classes of the Human Error Taxonomy relevant to the requirements engineering process?
- RQ2b: Do software developers believe that the Human Errors Taxonomy is useful for abstracting and classifying requirements engineering human errors?

Data analysis and results for RQ2a is discussed in subsection 4.3.2.1 and the data analysis for RQ2b is discussed in subsection 4.3.2.2.

4.3.2.1. RQ2a: Are all three Error types (slips, lapses, mistakes) and all error classes of the Human Error Taxonomy relevant to the requirements engineering process?

Data gathered during Study 2 was analyzed in order to answer this research question. During Study 2, each team abstracted human errors from faults in their own requirements document (i.e., SRS document). The error abstraction data was analyzed at two levels:

- Error Type Level: The high level error types of HET are slips, lapses, and mistakes. Analysis shown in Figure 12 indicates that all six teams made errors of the three types. That is, teams committed all three types of errors (slips, lapses, and mistakes) and these errors caused injection of faults in their SRS documents.
- Error Class Level: Each high-level error type of HET has some low-level error classes. Error types Slip, Lapse, and Mistake have two, two, and eleven classes, respectively. Thus, there are a total of 15 low-level error classes in HET. Analysis showed that eleven of the fifteen error classes were represented in the SRS documents of the teams. That is, teams made faults that were classified into most (but not all fifteen) of HET's error classes.

Overall, this analysis indicates that all three error types (slips, lapses, mistakes) in the HET are important and relevant because software developers made errors and faults of each type.

Results also showed that developers made faults that were abstracted to (or traced back to) errors belonging to eleven of the fifteen error classes in HET. Hence, HET's errors classes are also relevant to requirements engineering process, but need to be further studied so that more conclusions can be drawn about relevance of all fifteen classes.

4.3.2.2. RQ3: Do software developers believe that the Human Errors Taxonomy is useful for abstracting and classifying requirements engineering human errors?

The data gathered from Study 1, Study 2, and Study 3 was analyzed to answer this research question. First during Study 1 (the control group study), a post-study survey was conducted. In the survey, the experimental group and the control group participants rated HET (experimental group) and Requirement Error Taxonomy or RET (control group) across various usefulness categories on a 5-point scale (ranging from "1 – not useful" to "5 – very useful"). A focus group discussion was also conducted in order to understand the problems faced by participants while using HET/RET to find faults. Only the feedback data collected from the post-study survey was analyzed to answer this research question (RQ3). The results (Table 16) showed that while both error taxonomies were rated favorably, HET received slightly better feedback in four out of the five usefulness categories. These four categories were usability (ease of use), usefulness, confidence that error classes in taxonomy represent real RE problems, and

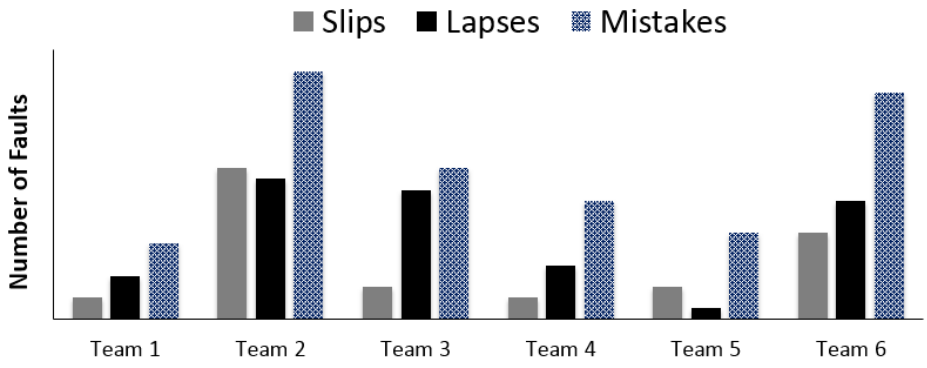


Figure 12. Study 2: Team Faults by Error Types

worthiness of effort spent in using the taxonomy. RET was rated more favorably for the category, “Error Classes Do Not Overlap”.

This was expected because unlike the RET, the HET includes errors within an error type (e.g., Application error – an error class under Mistake) that can happen at different points during the requirements development process (i.e., elicitation, analysis, and verification).

Table 16. Study 1: HET vs RET Comparison Using a 5-point Scale

	HET	**RET**
Usability of the taxonomy	2.9	2.6
Error classes in the taxonomy are distinct and do not overlap	2.9	3
Usefulness of the taxonomy	3.8	3.4
Confidence that error classes in taxonomy represent real requirements engineering problems	3.9	3.7
Worthiness of effort	3.3	3.2

Next, during Study 2 (feasibility study), participants rated the Human Error Taxonomy (HET) across nine specific characteristics: usefulness, intuitiveness, confidence, understandable, classification, abstraction, helpful, and no overlap of error classes. Table 17 provides the results of this analysis.

Table 17. Study 2: Post-Study Survey Results

HET Characteristic	Survey Statement	Mean Rating (on a 5-point Scale)
Usefulness	The HET is helpful for identifying faults	3.4
	The HET is complete	3.7
	The HET will be useful on future project	3.7
	The HET is helpful in improving the SRS	4.2
	The HET is helpful to detect overlooked faults	3.5
	The effort spent on using HET is valuable	4
Intuitiveness	The HET is intuitive	3
	I am confident that errors represent real problem	3.7
Confidence	I am confident in the error abstraction process	2.9
Understandability	HET is easy to understand	3.4
Classification	The HET is easy to use to abstract and classify errors	3
Helpful	The HET is helpful for understanding faults.	3.8
No overlap of errors classes	HET's error classes are distinct and do not overlap	2.9

Although, during Study 2, most of the HET-characteristics were rated positively, there were some characteristics that participants believed needed to be improved. Participants' feedback showed that they did not find the HET intuitive and they also were not confident about the error abstraction process. Additionally, participant feedback also showed that they did not

find HET easy to use when abstracting and classifying human errors from requirements faults. Overall, feedback from Study 2 participants revealed that the error abstraction process and the error abstraction training was not clear and needed to be improved.

Table 18. Study 3: Participants' Feedback about Educational Value of HET

Questions that evaluated effectiveness of HET and the error abstraction process for imparting knowledge of human errors				
		N	**Mean (SD)**	**Median**
Q1	I feel confident I can distinguish between a slip and a lapse	33	3.9 (0.6)	4
Q2	I feel confident I can distinguish between a slip and a mistake	33	4.1 (0.8)	4
Q3	HET documentation had sufficient detail to allow me to understand human errors that occur during the requirements development process	33	3.8 (1)	4
Questions related to educational value of human errors and the error abstraction process				
Q4	The effort spent in learning human errors is valuable and worthwhile in understanding faults in requirements document.	33	3.97 (0.9)	4
Q5	I am confident that human errors represent real problems in the requirements development process.	33	4.2 (0.8)	4

Study 3 specifically focused on the educational value of training students on the requirements engineering human errors described in the HET. During Study 3, the students were first trained on HET and the error abstraction process and then they were asked to abstract errors from 10 faults in an externally developed SRS document (PGCS SRS). Next, students were asked to provide feedback about, whether performing the error abstraction process (using HET) has helped them in understanding the difference between the three human error types (slips,

lapses, mistakes). With Study 3, the central idea was to evaluate if HET is a good learning resource for Computer Science and Software Engineering students to learn about Cognitive Psychology concepts (i.e., slips, lapses, mistakes). Table 18 provides the results of analyzing the feedback data collected during Study 3. As can be seen in Table 18, students believed that performing error abstractions (using HET) from faults helped them learn about slips, lapses, and mistakes in general and also the slips, lapses, and mistakes that occur during the requirements engineering process.

4.4. Summary of Results Obtained from the Three Studies

The three studies described in this chapter were designed to evaluate the feasibility of using the Human Error Taxonomy for requirements fault detection. The results from Studies 1 and 2 (shown in Figure 9, 10, and 11) showed that software developers were able to use human error information to find faults in requirements documents. The results (Figure 9 and 11) also showed that HET can provide improved fault detection effectiveness when compared to traditional inspection approaches like Requirements Error Taxonomy and Fault-checklist based inspections. The improvements, however, were not statistically significant. That is, during Studies 1 and 2, even though the inspectors who used HET found more faults, the HET did not help inspectors detect a *significantly larger number of faults* when compared to Requirements Error Taxonomy and Fault-checklist based inspections. Significance of results notwithstanding, the improved fault detection effectiveness provided by HET motivated further investigation of *using the HET to support requirements inspections*.

The post-study surveys and discussions conducted during the studies also revealed that study participants faced major difficulties when performing error abstraction from requirements faults. Error abstraction is an important leg of the HET-based inspection approach (called Error

Abstraction and Inspection or EAI). These results revealed that in order to improve the fault detection effectiveness of the HET-based inspection approach (i.e., the EAI approach), it is important to improve the training and tool support for the error abstraction leg of EAI.

To that end, I worked on developing an error abstraction tool called Human Error Abstraction Assist. The development of this tool was done under the supervision of a Cognitive Psychology expert, Dr. Gary Bradshaw (Professor, Mississippi State University). The next chapter provides a detailed description of the Human Error Abstraction Assist (HEAA) tool.

5. THE HUMAR ERROR ABSTRACTION ASSIST TOOL

EAI, which is the human error based (i.e., HET-based) inspection approach, begins with a fault checklist (FC) inspection step, which is followed by an error abstraction step, which in turn is followed by an error-informed re-inspection step. Results from Studies 1 and 2 (in Chapter 4) showed that, in order for improving the fault detection effectiveness of the HET-based inspection approach (*Error Abstraction and Inspection* or *EAI* approach), it was important that the error abstraction leg of EAI be improved. The error abstraction leg of EAI helps software development teams in identifying and understanding the human errors that were committed during the development process of a requirements document (these human errors in turn caused faults to be injected in the requirements document being inspected). Participants of the studies described in Chapter 4 stated in their feedback that the trainings and support they were provided on error abstraction were not sufficient for them to be able to accurately abstract errors from requirements faults. Participants faced considerable difficulties during the error abstraction step. This maybe because the error abstraction step requires inspectors to retrospectively analyze each fault (found during the fault-checklist inspection) to determine the cognitive process that went awry (or was flawed to begin with), thus causing the fault to be injected. Requirements development is a fuzzy process as it involves multiple activities (elicitation, analysis, documentation etc.) and multiple people (client, end-users, requirement analysts, requirement author). Hence, for an inspector, who may not have not been involved in the requirements development process, retrospectively analyzing a fault to determine the cognitive failure (human error) that caused the injection of the fault can be an overwhelming and complex task. Overwhelming, because the inspector can think of numerous scenarios where the cognitive failure might have occurred and this can cause difficulties for the inspector to pick the most likely scenario.

Table 19. Distribution of HET's Human Errors across RE Activities

RE Activities / Human Error Categories	Elicitation	Analysis	Specification	Management
Slips	Clerical Errors		Clerical Errors	
			Lack of consistency in Requirement Specifications	
Lapses	Loss of information from stakeholders			
	Accidentally overlooking requirements			
Mistakes	Application errors	Application errors		
	Environment errors	Environment errors	Environment errors	
				Information Management errors
	Wrong assumptions	Wrong assumptions		
	Low understanding of each other's roles	Low understanding of each other's roles		
	Mistaken belief that it is impossible to specify non-functional requirements in a verifiable form	Mistaken belief that it is impossible to specify non-functional requirements in a verifiable form		
	Not having a clear demarcation between client and users			
	Lack of awareness of sources of requirements			
		Problem-Solution errors		
				Inadequate Requirements Process
			Syntactic errors	

To alleviate this problem and assist the inspectors when they are trying to identify the human error that caused the injection of a fault, an intuitive questionnaire-style framework (Appendix B) was developed that helps inspectors accurately pinpoint the human error that caused the fault being analyzed. The Human Error Abstraction Assist (HEAA) works by guiding the inspector in eliminating the unlikely scenarios and focus on a scenario that is more likely to have caused the injection of the fault being analyzed.

1. Choose one of the following options to decide where the fault originated:

(a) Did the fault occur:

- While the system was being analyzed?
- While a large system was being divided into smaller parts?
- While system functionalities (functional requirements) and system behavior (performance and other non-functional requirements) were being determined?

(b) Did the fault occur during interviews or discussions with the stakeholders (end users, project sponsors, etc.)? This is where the user needs are gathered.

(c) Did the fault occur when the system information/requirements were being documented to create a formal software requirements document?

(d) Did the fault occur:

- During the *management* of the activities in a), b), or c) above?
- As requirements evolved or changed (i.e., traceability, version control, etc.)

RE activity associated to each option:

Option (a) – Requirement Analysis. **Option (b) –** Requirement Elicitation.
Option (c) – Requirement Specification. **Option (d) –** Requirement Management

Figure 13. Question# 1 in Human Error Abstraction Assist

The motivation behind HEAA's development was the belief that inspectors will be more comfortable if they focus their attention on requirements phase activities (elicitation, analysis, specification, and management) instead of focusing on understanding the mechanisms of human cognitive failures (i.e., slips, lapses, and mistakes). This is because inspectors are generally software developers and are expected to have more knowledge about the various requirements engineering activities as compared to being knowledgeable about slips, lapses, and mistakes. Therefore, for developing the HEAA, the fifteen human error classes in the Human Error Taxonomy were distributed across four major requirements engineering activities (elicitation, analysis, specification, and management). Table 19 on the pervious page provides the result of this distribution. The HEAA was developed based on this distribution of errors. A description of how the HEAA helps inspectors in mapping requirements faults to human errors is provided in subsection 5.1.

5.1. Error Abstraction Using HEAA

Abstracting human error from a given fault with HEAA begins with the inspector picking a requirements phase activity wherein the human error occurred and resulted in the injection of

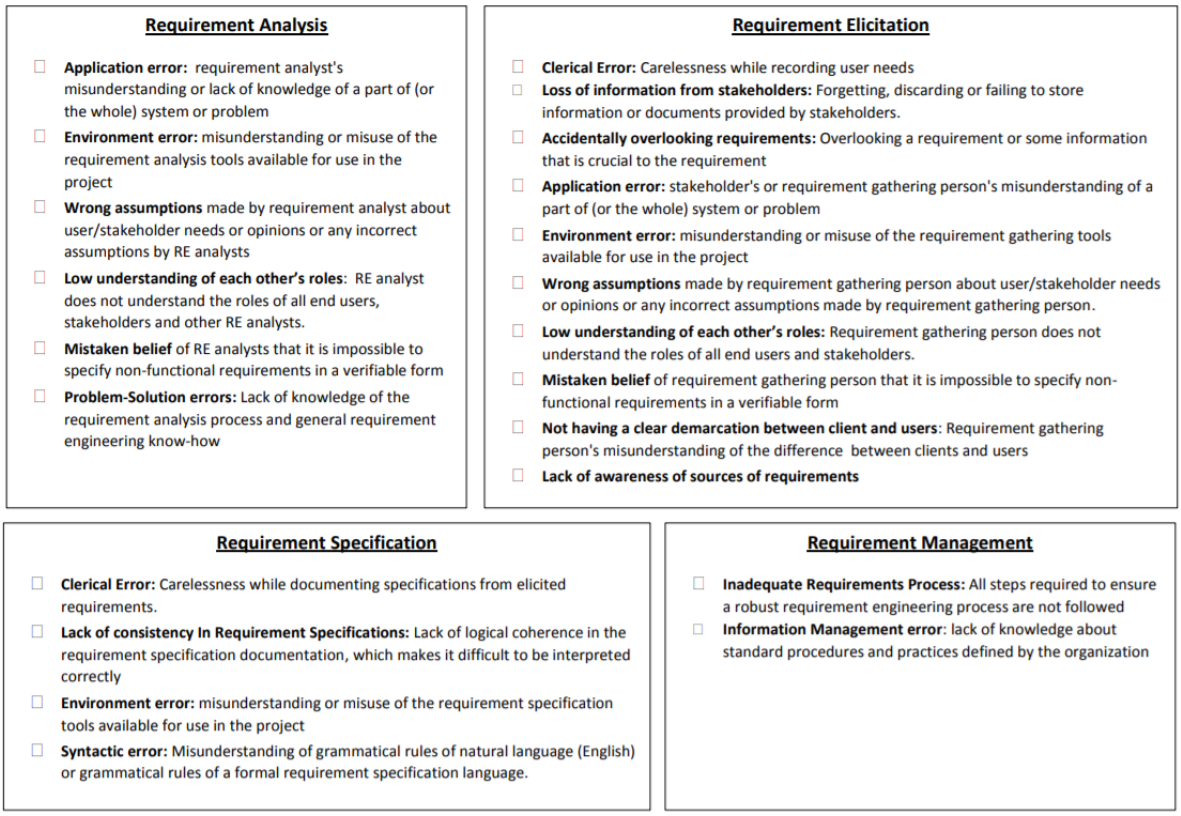


Figure 14. Question# 3 in Human Error Abstraction Assist

the fault being analyzed. As Question #1 in HEAA (see Figure 13), a checklist of items is provided to guide the selection of the requirements activity.

The next question (Question# 2) requires the inspector to visualize and provide an account of the scenario where the human error occurred. This helps the inspectors in improving their understanding of the requirement phase activity where the human error might have occurred (so in a sense, steps 1 and 2 are iterative in nature).

Next, as Question# 3, the inspector picks a human error from the options provided to him/her under error boxes labelled with requirement activity names.

Each box in Question # 3 (see Figure 14) is labeled with a particular requirements engineering activity and provides the human errors that are relevant to that requirements engineering activity. The boxes in Question# 3 of the HEAA tool were created based on the

distribution of human errors across requirements engineering activities (this distribution was shown in Table 19).

5.2. Evaluation of the Usefulness of HEAA Tool

The Human Error Abstraction Assist (HEAA) tool was created with the purpose of improving the error abstraction leg of the human error-based requirements inspection approach, Error Abstraction and Inspection (or EAI) approach. After the creation of the HEAA tool, empirical studies were conducted to evaluate whether the HEAA tool provides improved support (compared to Human Error Taxonomy) for software developers when they are trying to abstract human errors from requirements faults. It was anticipated that improved support during the error abstraction leg of EAI would improve the fault detection effectiveness of the EAI inspection approach. To that end, Chapter 6 described the controlled studies conducted to evaluate the usefulness of the Human Error Abstraction Assist tool during human error-based requirements inspections.

6. VALIDATION AND REFINEMENT OF THE HUMAN ERROR ABSTRACTION ASSIST TOOL

The creation of the Human Error Abstraction Assist (HEAA) tool was motivated by participant feedback that training and support for the error abstraction leg of the human error-based inspection approach (i.e., the Error Abstraction and Inspection or EAI approach) needed to be improved. After the creation of the HEAA tool, four empirical studies were designed and executed to evaluate its usefulness. The studies not only evaluated the usefulness of the HEAA tool during human error-based requirements inspections, but also evaluated the human error-based requirements inspection approach (i.e., the EAI approach) itself. Essentially, the four studies were a continuation of the series of studies (described in Chapter 4) to evaluate the usefulness of the human errors identified in Human Error Taxonomy, with the only exception being that the EAI inspection approach was now being supported by the newly developed HEAA tool. Because the four new studies are continuation in the series of empirical evaluations of human error-based requirements inspections, the four new studies are referred to as Studies 4, 5, 6, and 7 (the first three studies of the series were described in Chapter 4).

This chapter describes the procedure followed and the results obtained from the four empirical studies that were conducted after the creation of the HEAA tool. Section 6.1 provides the research questions that drove the designs of the four studies, followed by Section 6.2 that provides the study designs for the four studies. Section 6.3 provides the results obtained from the four studies.

6.1. Research Questions

Table 20 provides the research questions that were formulated to enable data collection for evaluating the usefulness of the human error-based requirements inspection approach (i.e., EAI approach) supported by the newly developed Human Error Abstraction Assist tool.

Table 20. Research Questions to Evaluate the Usefulness of the EAI approach when supported by the HEAA tool

#	Research Question
RQ1	Can the Error Abstraction and Inspection approach (supported by the Human Error Abstraction Assist tool) improve the fault detection effectiveness of inspectors when compared to traditional requirements inspection approach?
RQ2	Does the Human Error Abstraction Assist tool provide a useful method for abstracting human errors from requirements faults?
RQ3	Can error abstraction using the Human Error Abstraction Assist tool provide significant insights into the type of human errors that are committed most frequently during the requirements development process?

6.2. Description of Designs of the Four Empirical Studies

This section provides the designs of the four controlled studies that provided data for answering the research questions described in Table 20. This section provides the designs of the four studies, followed by Chapter 6.3 that provides the data analysis and results (based on the research questions shown in Table 20). As the four studies described in this chapter are continuation of the series of empirical evaluations (described in Chapter 4) of the usefulness of human error-based requirements inspections, the four studies are titled Studies 4, 5, 6, and 7.

6.2.1. Experiment Design for Study 4

The primary goal of Study 4 was to evaluate if the *Error Abstraction and Inspection* (EAI) approach supported by the Human Error Abstraction Assist tool will help inspectors

discover a significantly larger number of faults that are otherwise left undetected when using the standard fault checklist-based inspection approach.

The participants in Study 4 were 17 graduate students enrolled in the Software Requirements Definition and Analysis course at North Dakota State University. The participants were a mix of MS and PhD students in computer science or software engineering and had prior Information Technology (IT) industry experience. The course trained students on identifying, analyzing, documenting and verifying requirements.

Study 4 utilized two different requirements artifacts. During the initial training, participants performed a practice inspection using EAI on a software requirements specification (SRS) document that specified requirements for a Parking Garage Control System (PGCS). PGCS SRS described requirements for controlling the entries and exits of a parking garage. The PGCS SRS was 10 pages long and seeded by its original developers with 30 realistic faults. PGCS SRS was chosen for the training due to its generic domain and seeded set of faults. For the transfer session, participants used EAI to inspect the SRS document for Restaurant Interactive Menu (RIM) system. The RIM system is responsible for taking customer's orders in a restaurant with the help of an interactive PDA or online system. The RIM SRS was developed for a real project through interaction with clients, was 21 pages long, and contained real faults.

Figure 15 shows the experimental procedure followed during Study 4. The study was conducted in two phases: an initial training and the transfer session. The initial training consisted of two training sessions: Training 1 on fault checklist (FC) inspection approach, and Training 2 on EAI approach. Transfer session refers to the part of the experiment wherein participants apply (or transfer) the knowledge gained during the trainings to carry out the actual experimental tasks. The details of the trainings and transfer session steps are provided in Table 21.

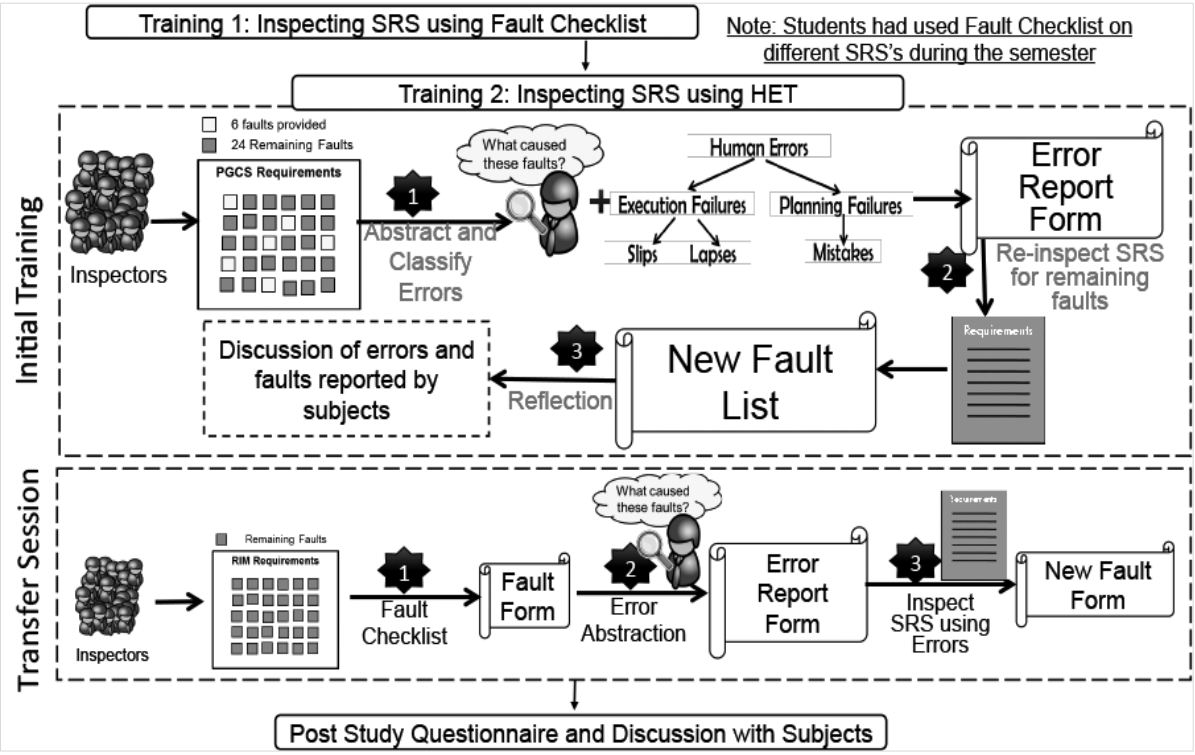


Figure 15. Study 4: Experimental Procedure

6.2.2. Improving the Human Error Abstraction Assist Tool

Using lessons learned from the previous study (i.e., Study 4), I first worked on refining/improving the Human Error Abstraction Assist (HEAA) tool, and next Study 5 was conducted wherein the Error Abstraction and Inspection (EAI) approach was supported by the refined-HEAA tool. Before describing the experiment design of Study 5, in this section a brief description of the improvements added to the HEAA tool is provided.

The goal was to improve the HEAA tool on several levels. First goal was to refine the HEAA to help the inspectors better visualize the situation/scenario wherein the human error occurred and led to the injection of the fault that is being analyzed. Second goal was to provide inspectors with a better view of HET's three human error types (slips, lapses, mistakes).

Table 21. Study 4: Steps Performed by Participants

Experimental Step		Description
Training 1 – Pre-experimental training on Fault-Checklist (FC) inspection.		Over the course of the semester, the participants have been trained on applying the fault-checklist (FC) approach on various SRS documents. Training 1 consisted of a quick recap session of the FC inspection approach.
Training 2 (PGCS SRS)	Initial training on EAI	This training was a 90-minute training session wherein the participants were trained on human error taxonomy (HET), on using the Human Error Abstraction Assist (HEAA) to abstract and classify human errors, and on using the abstracted human errors to find additional faults in the SRS document. After the participants were introduced to HET and HEAA, they were provided with the PGCS SRS and 6 (out of 30 seeded) randomly chosen faults.
	Step 1 – Error abstraction and classification on the 6 faults	The participants used information provided during initial training (on EAI) to abstract and classify human errors from the six faults using Human Error Abstraction Assist (HEAA). The output of this step was 17 individual Error Report Forms (one per participant) containing human errors present in PGCS SRS.
	Step 2 – Error-informed re-inspection of PGCS SRS for the remaining faults	The participants then re-inspected the PGCS SRS using the human error information contained in error report form (from Step 1). The output of this step was 17 individual New-Fault lists (one per participant) containing new faults found during the re-inspection. Following the completion of Step 2, the researchers discussed the issues faced by the participants when performing error abstraction, and re-inspection using the EAI process.
Transfer Session (RIM SRS)	Step 1 – FC Based inspection	The participants used the fault-checklist (FC) inspection approach to inspect the RIM SRS. This step resulted in 17 individual Fault forms (one per participant) containing faults present in RIM SRS.
	Step 2 – Human error abstraction and classification	Participants used the Human Error Abstraction Assist (HEAA) to abstract and classify human errors for each fault they found during Step 1. The result of this step was 17 individual Error Report forms containing human errors committed during the development of RIM SRS.
	Step 3 – Error-informed re-inspection of RIM SRS for remaining faults	The participants re-inspected RIM SRS using the human error information from Error Report forms (from Step 2). The output of this step was 17 individual New-Fault List forms containing new faults in RIM SRS (i.e., faults that were not found during FC inspection or Step 1).
Post Study Questionnaire		After completing the steps described above, participants provided feedback regarding the usefulness of EAI, HEAA and the training procedures. This feedback was required in order to better understand the results and make improvements in both the EAI and the HEAA tool.

In order to achieve these goals, a decision flow diagram (Figure 16) was added to the HEAA tool in consultation with the Psychology expert. The improved HEAA tool can be found in Appendix C. The decision flow diagram asks intuitive questions at the decision nodes, which helps the inspectors in selecting the right human error type for the fault they are analyzing.

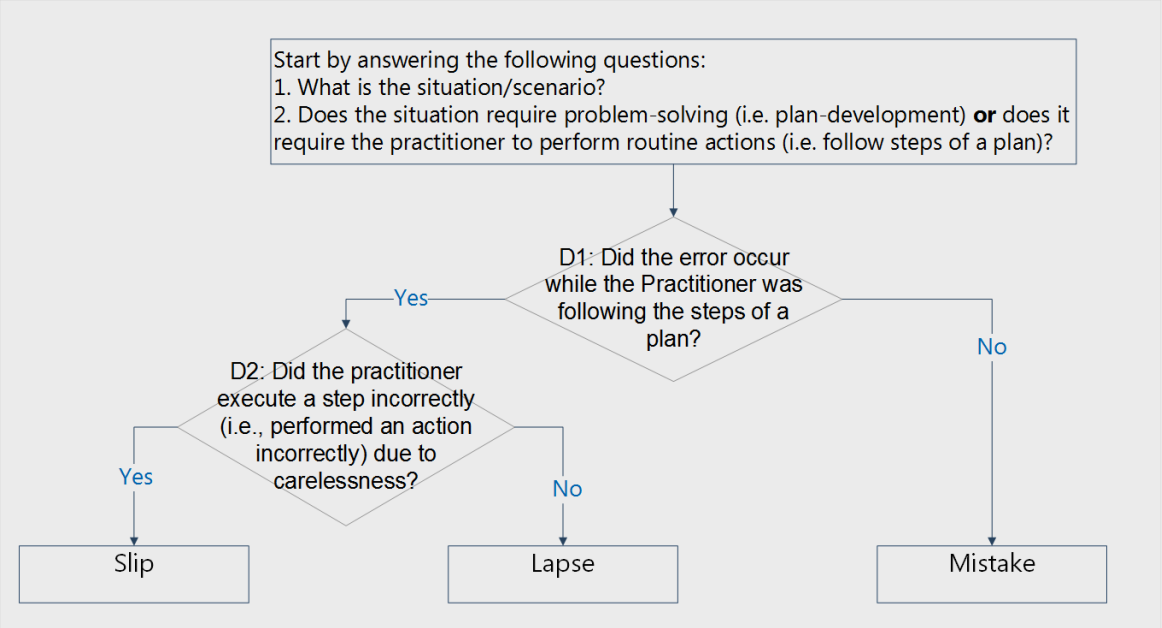


Figure 16. Decision Tree to Select Error Type

A noticeable change in the improved HEAA compared to the one used in Study 4 (see Chapter 5 for a description of the HEAA tool used in Study 4) is that the high level error types of HET (i.e., slips, lapses, mistakes) have been brought back into the fold. That is, the improved HEAA will require inspectors to choose an appropriate error type (slips/lapse/mistake) before they select a human error class. The inclusion of error type level in HEAA was suggested by the Cognitive Psychology expert (Dr. Gary Bradshaw). The rationale is that, from an inspector's perspective, an understanding of slips, lapses, and mistakes will promote his/her understanding of the detailed human error classes that are included in the HET. Concerning the above-mentioned inclusion of the error type level in HEAA, an additional module to the error abstraction training has been added. The additional module focuses on helping inspectors understand the difference between slips, lapses, and mistakes by demonstrating the process of mapping simple real world mishaps to slips, lapses, and mistakes. It will be demonstrated to the inspectors that the decision tree (Figure 16) can be used to categorize day-to-day accidents or

mishaps (such as clinical misdiagnosis by a doctor) as either a slip, lapse, or a mistake. Such examples are expected to help inspectors in understanding and using the refined HEAA tool (to map requirements faults to human errors) more effectively.

Furthermore, in order to improve the error abstraction training, more examples of the "fault to error-class mapping process" were needed. In order to do this, the training was updated to include demonstrations of the steps of error abstraction (using HEAA) for real faults in the RIM SRS. That is, for a selected set of RIM SRS faults, the training will include all steps, starting from situation/scenario formation, to error type selection (using decision flow diagram), to error-class selection.

Study 5, which is described in the next section, evaluated the EAI inspection approach supported by the refined-HEAA tool.

6.2.3. Experiment Design for Study 5

Similar to Study 4, the main goal of Study 5 was to evaluate if the Error Abstraction and Inspection (EAI) approach supported by the HEAA tool can help inspectors identify requirements faults that are overlooked or are hard-to-locate during traditional requirements inspections. One major difference between Study 4 and Study 5 was that Study 5 used the refined HEAA tool. Another primary goal of Study 5 was to gather and analyze data about: (1) issues that inspectors face when abstracting human errors from requirements faults and, (2) strategies that inspectors use when using the abstracted human error information to locate additional related faults.

Fifteen (15) Graduate students enrolled in the *Software Development Processes* course at North Dakota State University (NDSU) participated in Study 5. The course is a breadth course on software engineering topics and covers the entire software development lifecycle.

An artifact that described the requirements for a Restaurant Interactive Menu (RIM) system and contained naturally occurring faults was used during the study. The RIM system allows restaurant owners to control the inventory, and restaurant customers to order and pay bills. RIM SRS was used during Study 5 to enable comparison of the results with Study 4 (Study 4 also used RIM and evaluated the usefulness of the EAI inspection approach, but in Study 5 EAI was supported by the refined-HEAA).

The experimental procedure was carefully designed to gather insights that can be used to improve the Error Abstraction (EA) and the error-informed re-inspection steps of the EAI approach. Table 22 provides the descriptions of the procedure followed during Study 5.

Table 22. Study 5: Steps Performed by Participants

Experimental Step	Description
Training 1	During this 50-minute session, participants were trained on human errors in the Human Error Taxonomy, abstracting human errors from requirements faults using the refined-HEAA.
Step 1- Error Abstraction from RIM SRS faults	Participants were supplied with the RIM SRS and 16 known faults in the RIM SRS and were asked to abstract human errors from the 16 given faults. The output of this step was 15 error report forms (one per participant) containing human errors in RIM SRS. The output of this step helped in comparing the error abstraction results of all participants on same set of faults.
Step 2 – Error-informed inspection of RIM SRS	Participants were given the expected error abstraction results for each of the 16 faults. The expected abstraction results were decided in consultation with a Cognitive Psychologist, Dr. Gary Bradshaw. The participants were asked to use the provided human error information to inspect RIM SRS and locate additional faults related to the provided human errors. The idea behind giving participants the expected error abstraction results for each of the 16 given faults was to understand how individual participants use the *same human error information* to find additional related faults. The outcome of this step was 15 individual Fault Report Forms containing new faults in RIM SRS. The motivation behind asking participants to perform this task (i.e., error-informed inspection) was to evaluate if human error information helps inspectors in identifying additional faults (that are overlooked when only the traditional fault-checklist inspection approach is used).

6.2.4. Experiment Design for Study 6 (Live Study in a Conference)

Studies 5 and 6 were performed simultaneously, and while Study 5 evaluated the refined-HEAA tool in academic settings, Study 6 was targeted towards requirements engineering practitioners and researchers. Study 6 was performed at a requirements engineering (RE) conference called *Requirements Engineering: Foundations for Software Quality* (REFSQ) [48].

#	Fault Description	Error Description
1	***Information about the fault***: There are four primary data objects in the software for the Parking Garage Control System (PGCS): • k = maximum number of parking spaces in the parking garage • r = number of reserved parking spaces • a = number of non-reserved parking spaces that are available • o = number of occupied non-reserved parking spaces The system description section states that the number of reserved parking spaces (r) should not be higher than 40% of the maximum number of parking spaces (k). Functional Requirement# 7 (FR7) states that purchasing a monthly access card (which reserves a parking space), increases the value of r by 1. However, before the value of r is updated, the system needs to check the following condition: Is r less than 40% of k? FR7 does not perform this check. ***Fault***: FR7 is missing the following test: What if "r" > 0.4*"k"? ***Fault Location***: Line#170, Req#FR7	**Q1: RE activity in which the human error occurred (*pick one*):** ☐Analysis ☐Elicitation ☐Specification ☐Management **Q2: Human error type (pick one)** ☐Slip ☐Lapse ☐Mistake **What additional/specific background information would have helped you decide the type of error:**

Figure 17. Sample Error Report Form

The major goal of Study 6 was to evaluate if requirements engineering professionals are able to use the Human Error Abstraction Assist tool to abstract human errors from requirements faults. The population of interest were professionals with understanding of requirements engineering activities and industry experience. Participants were recruited at the venue of the conference (REFSQ conference). A total of 15 conference attendees volunteered to participate in Study 6. Although any background information regarding participants' experience was not gathered, the participants were a good mix of academic requirements engineering researchers (university professors) and industry practitioners (in software development organizations across world).

The requirements document used during the study was the document that specified requirements for a Parking Garage Control System (PGCS). Due to time restriction, instead of asking participants to read the entire PGCS requirements document, an error report form was prepared that provided background information and fault descriptions of 10 randomly selected faults in the PGCS requirements document. The participants were asked to abstract human errors from the 10 faults provided to them. The following supplementary documents were provided during the study:

- PGCS SRS: A printed copy of PGCS requirements document in case if participants wanted more background information related to the fault being analyzed.
- Refined-HEAA decision tree: A printed copy of the refined-HEAA tool was provided to help participants in abstracting human errors from the 10 given faults.
- Error Report Form: The error report form contained 10 faults in PGCS SRS. Participants were asked to abstract errors from the faults after they were provided a training on how to use the HEAA tool. Figure 17 provides the error reporting template for one of the 10 PGCS faults. Note that the participants used the HEAA tool to abstract only the requirements engineering activity and the error type (slip/lapse/mistake) for each fault. Due to time constraints, the participants were not asked to abstract the human error class for the faults.

Table 23 provides the procedure followed during Study 6.

Table 23. Study 6: Steps Performed by Participants

Experimental Step	**Description**
Error Abstraction Training	During a 30-minute session, participants were trained on the human error classes in Human Error Taxonomy, and how to use the refined-HEAA tool to abstract errors from the given faults.
Step 2 - Error abstraction and classification	Participants used the HEAA tool to abstract and classify human errors (into Slips, Lapses, and Mistakes) from 10 given faults in PGCS SRS.
Step 3 - Discussion of Results	The completion of error abstraction step was followed by a discussion of participants' results. The discussion step helped in gaining insights into the thought process of participants when they were analyzing faults and tracing the faults to human errors.

6.2.5. Experiment Design for Study 7

The primary goal of Study 7 was to evaluate the usefulness of Human Error Abstraction Assist (HEAA) in helping software developers in understanding/ identifying the human errors committed during the requirements engineering process. During Study 7, the objective was to evaluate the usefulness of HEAA for two distinct situations:

- Usefulness of the HEAA when identifying the human errors that were committed during the creation process of an externally-developed requirements document (note that the human errors, in this case, were committed by someone else).
- Usefulness of the HEAA when identifying the human errors that were committed during the creation process of a self-developed requirements document. Essentially, the idea was to evaluate whether HEAA can be used by software developers to map the faults (in their own requirements documents) to human errors that caused the injection of the faults.

Compared to the error abstraction training in Study 6, a small improvement was made to the error abstraction training during Study 7. A training supplement that provides the fault to error-class mappings for all the faults in PGCS SRS has been made available for helping inspectors in understanding the error abstraction process better. This training supplement was used during Study 7 during the Reflection step (shown in Figure 18 and described in Table 24).

Thirty-six (36) undergraduate computer science students enrolled in the Principles of Software Engineering course at North Dakota State University participated in this study. The course required students to work in teams to develop Software Requirements Specifications (SRS) for different software systems. After developing the SRS, the teams proceeded to implement the requirements. In this study, the focus was specifically on the faults and errors

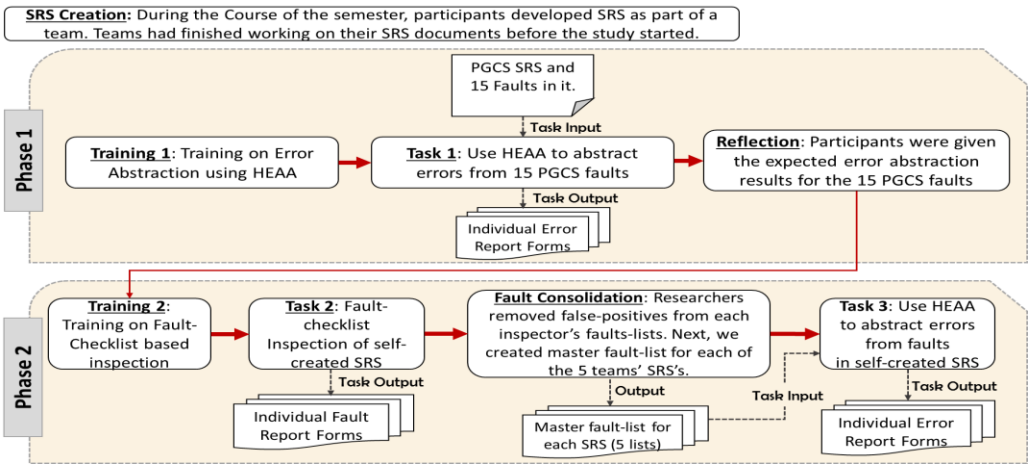


Figure 18. Study 7: Experimental Procedure

committed during the requirements (i.e., SRS) creation process. The students were divided into teams by the course instructor prior to this study.

Table 24. Study 7: Steps Performed by Participants

Experimental Step	**Description**
SRS Creation	The participants worked as part of teams to create requirements documents for different software systems. A description of the different software systems for which the teams created requirements documents, is provided in Appendix D. Note that teams had already created their requirement documents (i.e., their SRS's) before the study started.
Training 1 – Error Abstraction Training	During a 50-minute session, participants were trained on Human Error Taxonomy (HET), and on using the HEAA tool to abstract errors from faults. In order to demonstrate the HEAA tool's step-by-step procedure of mapping requirements faults to human errors, some generic requirements faults were used.
Task 1 – Abstraction and Classification of Human Errors in PGCS SRS	After the error abstraction training, participants were supplied with PGCS SRS, and 15 faults in the PGCS SRS. The participants were asked to use HEAA in order to map each of the 15 faults to human error that caused the fault. The outcome of this step was 36 individual error report forms (one per participant) containing human errors present in PGCS SRS.
Reflection	In this step, the participants were provided the expected error abstraction results for each of the 15 PGCS faults. The expected error abstraction results were obtained through discussions with a Cognitive Psychology expert (Dr. Bradshaw). The idea behind reflection step was to improve participants' understanding of the fault-to-error mapping process (i.e., the error abstraction process using HEAA).
Training 2 – Training on Fault-checklist Inspection Technique	The participants were trained on how to use the fault-checklist inspection technique to inspect SRS documents. This step is required to identify faults in a requirements document. The identified faults can then be mapped to human errors. So, essentially this step is required to initiate the error discovery process.
Task 2 – Fault-checklist Inspection of Self-created SRS	Participants inspected their self-developed SRS documents during this step. Each participant inspected the document they had created as part of their team. So, a participant who was part of Team 1 (see Appendix D) and created the SRS document for Dissertation Calculator system inspected the Dissertation Calculator SRS. The output of this task was 36 individual Fault Report Forms containing faults in different SRS documents created by the teams.
Fault Consolidation	This was performed by the researchers. As an example, for Team 1, I first compiled all the faults reported by the 8 team members. Next, I removed any false-positives (i.e., non-faults) and created a Master Fault List that only consisted of actual faults (true-positives) in Team 1's SRS. This was done for all 5 teams.
Task 3 - Abstraction and Classification of Human Errors in Self-created SRS	Participants were provided with the Master Fault List that was created for their SRS document (in previous step) and asked to individually abstract human errors (using HEAA) for each fault in their self-developed SRS's Master Fault List. The outcome of this step was 36 individual Error Report Forms containing human errors in the SRS documents created by the participants.

Study 7's objective was to evaluate the usefulness of HEAA, both when abstracting errors from faults in an *externally-developed SRS* and when abstracting faults in a *self-developed SRS*. Therefore, the study was conducted across two phases (see Figure 18). During Phase 1, an externally developed SRS that specified requirements for a Parking Garage Control System (PGCS) was used. During Phase 2 (see Figure 18) of the study, participants abstracted human errors from faults in the SRS documents that they had developed (as part of a team) during the course of the semester. Appendix D provides a description of the systems for which SRS's were created by each team. Table 24 provides the steps performed by the participants during Study 7.

6.3. Analysis of Data Gathered During Studies 4, 5, 6, and 7

This section provides the results of analyzing the data gathered during Studies 4, 5, 6, and 7. This section is organized around the three Research Questions (shown in Table 20, Section 6.1) that were formulated to validate the usefulness of the Error Abstraction and Inspection (EAI) approach supported by the Human Error Abstraction Assist (HEAA) tool. The four studies (Studies 4, 5, 6 and 7) were designed with the goal of collecting the data to answer the three Research Questions described in Table 20. Experimental design for each of the four studies were provided in Sections 6.2.1, 6.2.3, 6.2.4 and 6.2.5, respectively.

6.3.1. RQ1: Can the Error Abstraction and Inspection Approach (supported by the Human Error Abstraction Assist Tool) Improve the Fault Detection Effectiveness of Inspectors when Compared to Traditional Requirements Inspection Approach?

Data gathered during Studies 4 and 5 was analyzed to answer this research question.

During Study 4, participants first detected faults in a requirements document (RIM SRS) using the fault-checklist inspection technique. Next, they used the HEAA tool to abstract human errors from the faults they had found during the fault-checklist inspection. Finally, the

participants performed an error-informed reinspection on the RIM SRS. It was found that during the fault-checklist inspection of RIM SRS, participants found an average of 6 faults. Whereas, during the error-informed reinspection of RIM SRS, participants were able to locate an average of 14 new faults that they were not able to find during the first inspection (i.e., the fault-checklist inspection). Figure 19 shows the result of this analysis. Figure 19 compares the fault count of each participant during fault-checklist inspection (bottom portion of each column) vs. the new fault count during re-inspection using EAI (top portion of the same column). For example, participant S1 (inspector# 1) found 3 faults during the first inspection (using fault-checklist), and found 7 new faults during the re-inspection (using EAI), which computes to a percentage increase of 233% in fault detection effectiveness. Overall, participants found an average of 6 faults during the first inspection (fault-checklist) and an average of 14 new faults during the second inspection (EAI), with an average increase in effectiveness of 225%. These results provide evidence that an error-abstraction and inspection (EAI), supported by HEAA, can help inspectors discover a significantly more number of faults in an SRS that are otherwise left

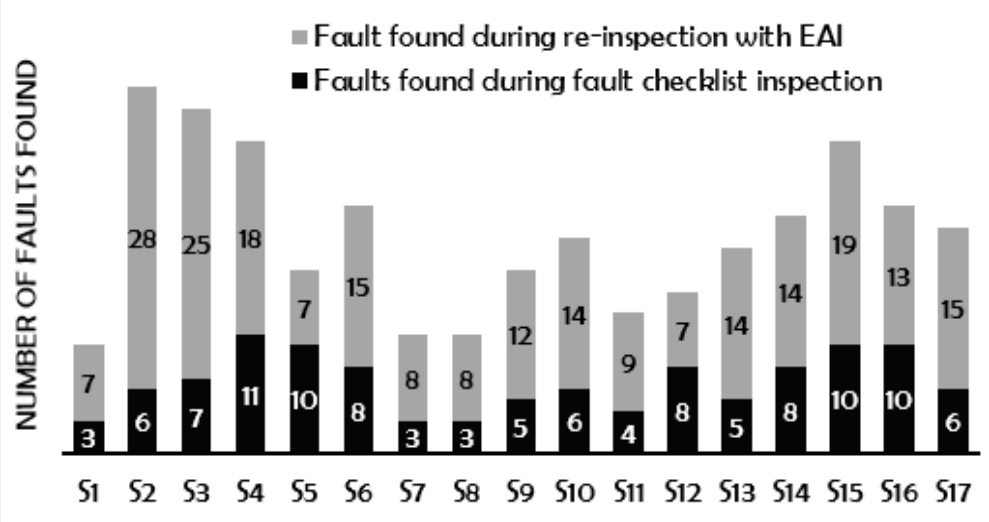


Figure 19. Study 4: Effectiveness of EAI vs. Fault-checklist inspection

undetected during the fault-checklist based inspection. The result of the one-sample test ($p<0.001$) showed that the average number of faults found using EAI (14) was significantly higher than average number of faults during FC inspection (6). Furthermore, even though the participants were re-inspecting the same document during EAI-based reinspection, the significantly large number of additional faults found shows that EAI is a very useful addition to FC for improving requirements quality.

During Study 5, participants were given 16 faults in the RIM SRS document and were asked to use the HEAA tool to abstract errors from the given faults. Next, the participants were provided with human errors that caused the injection of each of the 16 faults (for reflection purpose). The participants then used human error information to inspect the RIM SRS document. As shown in Figure 20, all participants were able to use human error information to find at least some new faults in RIM SRS (the new faults were not part of the list of 16 faults given to them). On an average, participants in Study 5 found *11.4* new faults in RIM SRS. A one-sample t-test showed the mean number of faults found by participants using error information was significantly larger than zero ($p<0.001$).

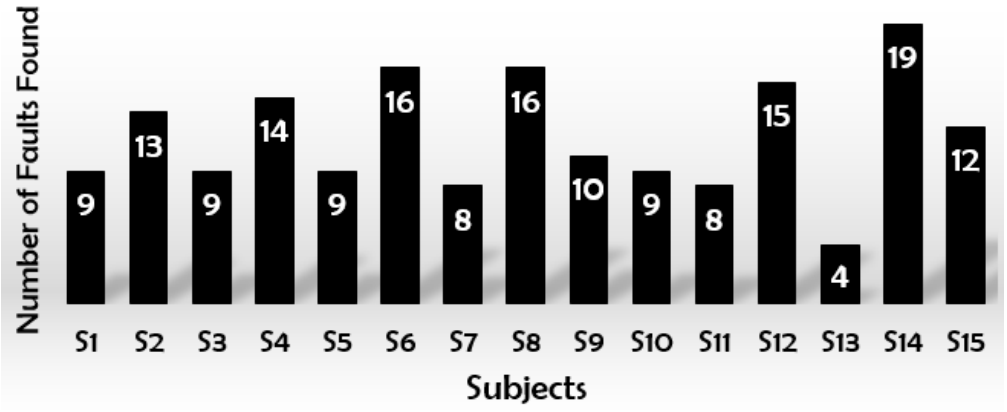


Figure 20. Study 5: Number of New faults Found During Error-informed Inspection

The analyses from Studies 4 and 5 shown above (in Figures 19 and 20, respectively) provide evidence that the *Error Abstraction and Inspection (EAI) approach when supported by the Human Error Abstraction Assist tool* can improve the fault detection effectiveness of inspectors when used in conjunction with the traditional fault-checklist inspection approach.

Another analysis - *related to gathering insights about what strategies are used by inspectors during the error informed reinspection step of EAI approach* - was performed on data collected during Study 5. During Study 5, participants were provided with 16 faults in RIM SRS documents and they were also provided with the human errors that caused the injection of the given 16 faults. The participants were then asked to use the error information to find related faults in RIM SRS. The idea behind supplying all participants with the error information was to examine: *When provided with the correct human error (that caused a fault), where in the SRS document do participants look in order to find other related faults that were caused by the human error*?

An interpretive analysis (see Table 25) was performed on the fault data provided by participants during the error-informed inspection of RIM SRS (during Study 5).

The goal of this interpretive analysis was to obtain insights into how participants make use of error-information to find additional faults that are related to the errors. Currently, the error-informed inspection is an ad-hoc process, wherein individual inspectors devise their own strategies to locate new faults.

We examined the location of all the new faults reported by the participants (not just the true-positives) for each of the 16 fault-error combinations provided to them.

Table 25. Study 5: Strategies Used by Inspectors during Error-informed Reinspection

Strategy Title	Strategy Description	Example
Additional faults in the same requirement	Participants reviewed the requirement that contained the original fault when looking for additional faults. The rationale was that the same requirement may contain more faults (not necessarily similar to original fault) because the requirements engineer/s who worked on creating the requirement were already under the influence of the human error (that was abstracted from the given original fault)	As an example, for Fault #1, the error happened while specifying the requirement titled, 'RIM_CUSTLOGIN_S01'. This requirement can be found between lines Line 77 to 124 in the RIM SRS. Multiple participants used this strategy to report other faults in the same requirement. Furthermore, three participants successfully found a new fault (true-positive) on Line #105.
Additional similar faults in other similar requirements	Participants reviewed the RIM SRS to first find any requirements that were similar to the requirement in which the original given fault was located. Next, if participants were able to identify any similar requirement, they reviewed the identified requirement to find faults that were similar to the original fault. The rationale was that if a human error occurred while creating a specific type of requirement, then it is possible that the same human error might have occurred while creating other similar requirements.	As an example, Fault #2 is located in the requirement stated between lines 77-124, more specifically in the constraints section of the requirement. Fault #2 occurred due to carelessness while performing numerical calculations. Participants looked at constraints sections of other requirements, specifically where numerical calculations may have been performed by requirements engineer/s. Eleven (11) participants successfully used this strategy to find a similar calculation-related fault in a different requirement.
Additional faults in related requirements	The creators of RIM SRS have attached a related requirements section with every requirement. Participants reviewed requirements related to the original requirement (in which the original given fault was located). The rationale was that if a human error occurred during the creation of a requirement, then the human error might have affected the creation process of related requirements as well.	As an example, Fault #3 is in a requirement titled, 'RIM_REQUEST_HELP_S03'. Four participants reviewed a related requirement titled, 'RIM_ORDER_ENTREE_D08' and reported faults (found to be false-positives) in the latter requirement.
Additional similar faults in other requirements	For simplistic faults like 'missing information or missing words', participants just read through the whole SRS to find if there are other instances where requirement-sentences were missing or words in the sentence were missing (rendering the requirement incomplete or ambiguous).	As an example, Fault #8 states that "Hacker is listed in the list of actors. However, the hacker has no role in this requirement". Participants simply looked at the 'Actor' sub-section of all requirements to find if other requirements had faulty 'Actor' list. Four participants successfully found a fault on Line #346. The fault was that the 'Actor' sub-section of the requirement was left blank (whereas all use cases must have at least one actor).

Please recall that, during Study 5, for each of the 16 RIM SRS faults that were provided to the participants, they were also provided the human error that caused the fault. This analysis necessitated interpretation because, for each original given fault, we needed to compare and contrast the location of the reported faults (which participants deemed to be related to the original fault) with the location of the original fault. Furthermore, we had no quantitative data about: why a participant thought that a certain reported fault was related to the original fault and human error (that was provided for the original fault). There was also no quantitative data about what prompted the participant to look for a related fault at a particular location. Therefore, we needed to derive meaningful interpretations (from the locations of the reported faults) about how a participant (or multiple participants) found the particular reported faults. In other words, we were looking for prompts that participants created in their mind when looking for new faults. This analysis revealed four major strategies (shown in Table 25) that participants used in order to locate new faults related to the given fault-error combinations. In the future evaluations of the EAI inspection approach, the inspectors will be trained on these reinspection strategies.

6.3.2. RQ2: Does the Human Error Abstraction Assist Tool Provide a Useful Method for Abstracting Human Errors from Requirements Faults?

This research question focused on the usefulness of HEAA tool. A major focus was to evaluate if using the HEAA tool to support the EAI inspection approach works better when compared to simply using the Human Error Taxonomy (HET) to support the EAI approach (HET was used to support EAI during the first set of evaluations that were discussed in Chapter 4).

Another focus of this research question (RQ2) was to evaluate the refinements/improvements that were added to the HEAA tool using the lessons learned during the empirical studies. After its creation, the HEAA tool was first used during Study 4. After

Study 4, the HEAA was refined (see Section 6.2.2.) and the refined HEAA tool was used in the rest of the studies (Studies 5, 6, and 7). Hence, it was important to evaluate whether these refinements helped in improving the effectiveness of the HEAA tool.

Based on the discussion above, it can be seen that Research Question 2 (RQ2) is a multipart research question. Hence, RQ2 has been subdivided into the following research questions:

- RQ2a: Does the Human Error Abstraction Assist tool improve the error abstraction effectiveness of inspectors when compared to abstracting errors using Human Error Taxonomy?
- RQ2b: What insights into – the problems faced by the inspectors during the process of error abstraction using HEAA tool – can be used to improve the error abstraction process?
- RQ2c: Is the error abstraction process (supported by the HEAA tool) more effective when employed by professional requirements engineers as compared to when it is employed by students?
- RQ2d: Is the error abstraction process (supported by the HEAA tool) more effective when employed on self-developed requirements documents as compared to when employed on externally-developed requirements documents?

Data analyses and results for RQ2a, RQ2b, RQ2c, and RQ2d are discussed in subsections 6.3.2.1. 6.3.2.2, 6.3.2.3, and 6.3.2.4, respectively.

6.3.2.1. RQ2a: Does the Human Error Abstraction Assist tool improve the error abstraction effectiveness of inspectors when compared to abstracting errors using Human Error Taxonomy?

It is important to understand if the HEAA tool has improved the error abstraction process, which is an important step of the human error based requirements inspection approach (i.e., the Error Abstraction and Inspection or EAI approach). In order to measure the improvements provided by the HEAA tool, a metric called *Error Abstraction Accuracy* is used. Error abstraction accuracy is calculated for each inspector and it is simply the percentage of correctly abstracted errors out of the total number of errors reported by an inspector. As an example, if an inspector abstracted 16 human errors (from 16 requirements faults), and out of the 16 reported errors, 8 were correctly abstracted, then the *Error Abstraction Accuracy* for the inspector would be 50% (i.e., 8/16).

To answer the research question (RQ2a), a comparison was made between the error abstraction accuracies achieved by participants of Study 1 vs the participants of Study 4. During Study 1, twenty-three participants from the experimental group used the Human Error Taxonomy to abstract errors from 10 faults in the PGCS SRS (details about Study 1 can be found in Section 4.2.1). For all twenty-three Study 1 participants, the mean error abstraction accuracy (when abstracting and classifying human errors from the same 10 given PGCS SRS faults) was found to be 15.45%. That is, overall, the 23 participants were able to achieve an error abstraction accuracy of 15.45% (using the Human Error Taxonomy) when abstracting and classifying human errors from the 10 given PGCS SRS faults.

Now, during the first phase of Study 4, participants were supplied with 6 faults in the PGCS SRS and were asked to abstract human errors from the faults. But, unlike Study 1 (in

which participants were trained on error abstraction using Human Error Taxonomy), during Study 4 participants were trained on error abstraction using the HEAA tool (more details about Study 4 can be found in 6.2.1). It was found that participants in Study 4 were able to achieve a mean error abstraction accuracy of 26.04%. Table 26 compares the error abstraction accuracy that was achieved using Human Error Taxonomy in Study 1 vs the error abstraction accuracy that was achieved using Human Error Abstraction Assist in Study 4.

Table 26. Study 1 vs Study 4: Error Abstraction Accuracy Comparison

Study Description	Mean Error Abstraction Accuracy
Study 1: 23 participants used the Human Error Taxonomy to abstract human errors from faults in PGCS SRS.	15.45%
Study 4: 17 participants used the Human Error Abstract Assist tool to abstract human errors from faults in PGCS SRS.	26.04%

Recall that Study 4 had two phases: a training session and a transfer session. The error abstraction accuracies achieved by Study 4 participants during training session were shown in Table 26. Next, during Study 4's transfer session, participants used the HEAA tool to abstract errors from faults in the RIM SRS. The error abstraction accuracies achieved by participants during Study 4's transfer session were also analyzed. It was found that participants were able to achieve a mean error abstraction accuracy of 38% when abstracting errors (suing HEAA) from faults in RIM SRS. After completion of Study 4, the HEAA tool was improved/refined based on participant feedback and the refined HEAA tool was used during Study 5. During Study 5, participants were given 16 faults in the RIM SRS and were asked to use the refined HEAA tool

to abstract errors from the given 16 faults. The mean error abstraction accuracy achieved by participants of Study 5 was found to be 45%.

Table 27. Study 4 vs Study 5: Error Abstraction Accuracy Comparison

Study Description	Mean Error Abstraction Accuracy
Study 4: 17 participants used the Human Error Abstract Assist tool to abstract human errors from faults in RIM SRS.	38%
Study 5: 15 participants used the Human Error Abstract Assist tool to abstract human errors from faults in RIM SRS.	45%

The analyses presented in Table 26 and 27 show that:

- The Human Error Abstraction Assist tool has improved the error abstraction accuracy of inspectors when compared to Human Error Taxonomy.
- The improvements made to the HEAA tool (after Study 4) has helped in further improving the error abstraction accuracy of inspectors. A discussion on the improvements made to the HEAA tool was provided in Section 6.2.2.

6.3.2.2. RQ2b: What insights into – the problems faced by the inspectors during the process of error abstraction using HEAA tool – can be used to improve the error abstraction process?

Data gathered during Study 5 was analyzed to answer this research question. Study 5 was specifically designed to collect insights about the major issues inspectors face when using the HEAA tool to abstract errors from requirements faults. The HEAA tool used during Study 5 is provided in Appendix C. HEAA is a control flow style process, wherein control statements appear (inside decision nodes) in a top to bottom order. When using HEAA, inspectors have to make decisions at three (3) levels. At *Level 1*, the inspector has to decide the requirements

engineering activity in which the fault originated (i.e., the human error occurred). At *Level 2*, the inspector has to decide the high level human error type that was committed (slips/lapse/mistake). Based on decisions made for Levels 1 and 2, at *Level 3*, inspectors have to select an adequate human error class. For inspectors' convenience, HEAA's decision tree has been unpacked into a detailed, self-explanatory, and stepwise system that can be found in Appendix C. It is important to note here that, because HEAA is a control flow style process, if am inspector makes an incorrect decision in an initial decision level, then the rest of the flow is automatically rendered incorrect.

During Study 5, participants were given 16 faults in the RIM SRS document and were asked to use the refined HEAA tool to abstract errors from the faults. The output of this task helped in comparing the error abstraction results of all participants for same set of faults. The idea was to investigate the following: *At what level of the error abstraction process (i.e., when using HEAA) are participants making most of the misjudgments*?

Table 28 provides an overview of the error abstraction results reported by the 15 participants. Each row in Table 28 provides error abstraction accuracy at different HEAA levels across all the participants for each fault. As an example, for Fault #2, 13 out of 15 participants were able to select the correct requirements engineering activity (where the fault originated). Therefore, we only evaluated the rest of the abstraction data of those 13 participants who selected the correct requirements engineering activity. The analysis showed that, 11 of those 13 participants selected the right error type (slips/lapse/mistake). Furthermore, only 9 of the remaining 11 participants selected the correct human error class for Fault #2. Overall, 9 out of 15 participants provided the expected error abstraction result for Fault #2 (i.e., only 60% of

participants were able to accurately abstract the human error that caused Fault #2). Similar analysis was performed for all 16 faults.

Table 28. Study 5: Progressive Error Abstraction Correctness at the Three Decision Levels of HEAA

Fault #	Number of participants who chose the correct RE activity (Level 1 of HEAA)	Number of participants who chose the correct Error Type (Level 2 of HEAA)	Number of participants who chose the correct Error Class (Level 3 of HEAA)	Overall Correctness: Number of participants who reported correct EA result for the fault (correct at all 3 levels)
Fault 1	100% (15/15)	93.33% (14/15)	100% (14/14)	93.33% (14/15)
Fault 2	86.67% (13/15)	84.62% (11/13)	81.82% (9/11)	60% (9/15)
Fault 3	66.67% (10/15)	100% (10/10)	100% (10/10)	66.67% (10/15)
Fault 4	53.33% (8/15)	75% (6/8)	83.33% (5/6)	33.33% (5/15)
Fault 5	80% (12/15)	83.33% (10/12)	90% (9/10)	60% (9/15)
Fault 6	66.67% (10/15)	80% (8/10)	37.5% (3/8)	20% (3/15)
Fault 7	33.33% (5/15)	60% (3/5)	33.33% (1/3)	6.67% (1/15)
Fault 8	66.67% (10/15)	80% (8/10)	87.5% (7/8)	46.67% (7/15)
Fault 9	73.33% (11/15)	63.64% (7/11)	85.71% (6/7)	40% (6/15)
Fault 10	53.33% (8/15)	87.5% (7/8)	57.14% (4/7)	26.67% (4/15)
Fault 11	46.67% (7/15)	100% (7/7)	71.43% (5/7)	33.33% (5/15)
Fault 12	86.67% (13/15)	100% (13/13)	76.92% (10/13)	66.67% (10/15)
Fault 13	66.67% (10/15)	70% (7/10)	71.43% (5/7)	33.33% (5/15)
Fault 14	46.67% (7/15)	85.71% (6/7)	83.33% (5/6)	33.33% (5/15)
Fault 15	53.33% (8/15)	100% (8/8)	87.5% (7/8)	46.67% (7/15)
Fault 16	100% (15/15)	93.33% (14/15)	64.29% (9/14)	60% (9/15)

Figure 21 compares the error abstraction accuracies achieved by the participants at the 3 levels of HEAA. It was found that participants frequently misjudged the requirements engineering activity in which the faults originated. Furthermore, this analysis showed that, if participants picked the right requirements engineering activity, they were able to pick the correct

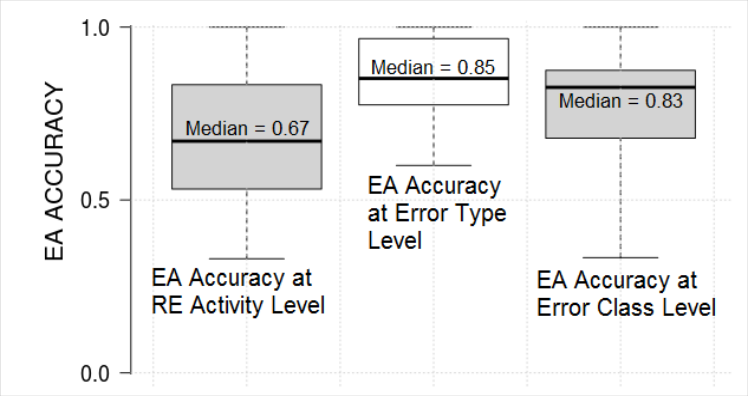


Figure 21. Study 5: Error Abstraction (EA) Accuracy at three HEAA Levels

Error Type (slips/lapse/mistake) and the correct Error Class in most cases. Figure 21 shows that the participants had the most difficulty when picking the adequate requirements engineering activity wherein the human error occurred (and resulted in the insertion of the fault being analyzed). Overall, for all 16 faults, participants achieved a median error abstraction accuracy of 67% at the requirements engineering activity level (compared to 85% at Error Type level and 83% at Error Class level). HEAA is a decision flow process and selecting an appropriate requirements engineering activity is the first decision that the inspectors have to make. Selecting the incorrect requirements engineering activity essentially renders the rest of the error abstraction effort futile.

The analysis provided above (in Table 28 and Figure 21) revealed that in order to improve the error abstraction accuracy of inspectors, the training on requirements engineering needs to be improved.

6.3.2.3. RQ2c: Is the error abstraction process (supported by the HEAA tool) more effective when employed by professional requirements engineers as compared to when it is employed by students?

Data gathered during Study 6 (Live study at a conference) was analyzed to answer this research question.

During Study 6, requirements engineering researchers and industry professionals were provided with 10 faults in the PGCS SRS and were asked to use HEAA to abstract errors from the given faults. Overall, the participants were able to achieve an error abstraction accuracy of 59%. The main goal of Study 6 was to gather feedback about the HEAA tool from requirements engineering professionals. The requirements engineering professionals provided the following comments during a discussion session:

- Participants stated that inspectors may have different understanding of requirements engineering activities (especially the order of analysis activity and specification activity) depending on the life-cycle employed in their software development projects. It was suggested that a glossary of requirements engineering activities should be provided (for future studies) to assist the inspectors during the error abstraction process.
- Participants stated that during the error abstraction training, providing definition and examples of a "RE specific plan" (as opposed to everyday plan failures) would help inform error analyses. The current error abstraction training provides example about how to map everyday failures (like pouring orange juice in cereal instead of milk) to slips/lapses/mistakes. The participants suggested that training should provide examples of the different types of requirements engineering plans

(e.g., requirements elicitation plan, requirements analysis plan, and requirements management plan). This will help inspectors in better visualizing the situations/scenarios in which human errors occur during the various requirements engineering activities.

6.3.2.4. RQ2d: Is the error abstraction process (supported by the HEAA tool) more effective when employed on self-developed requirements documents as compared to when employed on externally-developed requirements documents?

Data gathered during Study 7 was analyzed in order to answer this research question.

Study 7 was conducted across two phases: (1) Phase 1, in which participants abstracted errors from faults in an externally-developed PGCS SRS, and (2) Phase 2, in which the participants abstracted errors from the faults in the SRS documents that they had developed as part of their team.

Table 29. Study 7: Error Abstraction Accuracy when Abstracting Errors from Faults in Externally Developed SRS vs Faults in Self-Developed SRS

Team #	Mean Error Abstraction Accuracy when Abstracting Errors from Faults in Externally-developed SRS (PGCS SRS)	Mean Error Abstraction Accuracy when Abstracting Errors from Faults in Self-developed SRS
Team 1	35.36%	60.74%
Team 2	34.57%	60.09%
Team 3	32.14%	58.67%
Team 4	38.73%	58.12%
Team 5	39.52%	59.88%

Table 29 presents a comparison between the error abstraction accuracies achieved by the participants during Phase 1 vs Phase 2. In order to perform the analysis shown in Table 29, first

the individual error abstraction accuracy for each participant was calculated. Then, the mean accuracy for each team was calculated. It was necessary to calculate the mean error abstraction accuracy for each team (and not all participants together) because, during Phase 2 participants abstracted errors from faults in different SRS documents.

As can be seen in Table 29, participants in Study 7 achieved significantly higher error abstraction accuracies when using the HEAA tool to abstract errors from faults in their self-developed SRS documents.

6.3.3. RQ3: Can Error Abstraction Using the Human Error Abstraction Assist Tool Provide Significant Insights into the Type of Human Errors that are Committed Most Frequently During the Requirements Development Process?

Data gathered during Study 7 was analyzed to answer this research questions. During Study 7, participants worked as part of a team to develop requirements documents (i.e., SRS documents) for different software systems. There were a total of 5 teams in Study 7 that developed 5 SRS documents. One of the objectives of Study 7 was to understand what kind insights are generated when software development teams use the HEAA tool to abstract errors from faults they committed when creating their requirements documents. It was important to evaluate this because, the HEAA tool was developed to help software developers understand the human errors that they frequently commit when creating their requirements. A software development team's understanding of the most commonly occurring human errors during *their* requirements engineering process can help them in avoiding these human errors in future.

It was anticipated that abstracting errors from faults in an SRS document can generate tailored insights about the most common human error related issues that a team faced when creating their SRS document. That is, each requirements creation effort is different in that it

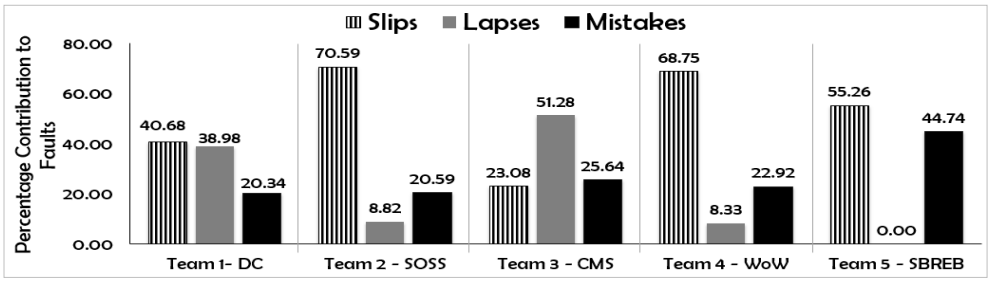


Figure 22. Study 7: Percentage Contribution of Slips, Lapses, and Mistakes to Faults

involves different personnel who are trying to create requirements for different software systems, and hence the human errors committed by them are also different. Hence, it can be worthwhile to identify the type of human errors different requirements engineering teams are more prone to.

First, I analyzed, for each team separately, the most frequently committed high level Error Types (Slip/Lapse/Mistake). Figure 22 shows the result of this analysis. As can be seen in Figure 22, the distribution of Slips, Lapses and Mistakes are different for different teams. This is because different teams worked under different environments and were trying to solve unique problems (the software systems they were creating the requirements for were different), and as a result the human errors they committed were also different. Figure 22 also shows that Slips were the leading cause of fault-injection when the teams were creating their requirements documents, followed by Mistakes. One clear trend that was seen across all five teams was that execution errors (Slips and Lapses together constitute execution errors) contributed to 60-80% of all the faults.

This result is consistent with Cognitive Psychology research, where researchers have shown that most of the errors committed by human operators are execution errors. Human subject based studies in Cognitive Psychology have shown that 60-70% of all detected human errors are execution errors [39, 49].

Table 30. Study 7: Percentage Contribution of Human Error Classes to Faults

	Slips		**Lapses**		**----------Mistakes----------**										
	Cler.	LC	LI	Accd.	Appln	Env	IM	WA	PU	MB	NH	LA	PS	IR	Synt
Team1	23.7%	17%	13.6%	25.4%	11.9%		1.7%	5%	1.7%						
Team2	55.9%	14.7%		8.9%	5.9%			8.8%				5.9%			
Team3	23%		15.4%	36%	12.8%		5.1%							7.7%	
Team4	68.8%			8.3%	6.3%	8.3%		4.2%						2%	2%
Team5	42.1%	13.2%				15.8%	18.4%							7.9%	

Next, in order to generate deeper insights about frequently committed human errors, an analysis was performed to examine the contribution of Human Error Taxonomy's human Error Classes (see Figure 7) to the faults in each team's SRS document. Table 30 provides the result of this analysis. Note that each row in Table 30 provides the distribution of human errors for one team. The sum total of each row in Table 30 is 100%. As can be seen in Table 30, teams committed different types of human errors when they were developing their requirements. For Team 1, the major cause of fault-injection was errors that happen due to Accidentally Overlooking Requirements (an error class under Lapse). Such errors happen when requirements engineers or end users or stakeholders forget to include a requirement or some information related to a requirement. Such overlooks are generally caused when end-users/stakeholders think that some things are obvious and fail to verbalize such information (a very common example is lack of requirements related of exception handling in SRS's). For Team 2, most of the faults were traced back to Clerical Errors, an error class under Slip error types. Clerical errors happen due to carelessness during mechanical transcription of requirements from one medium to another (for example, carelessness when creating formal requirements specifications from elicitation-notes). For Team 3, errors that happen due to Accidentally Overlooking Requirements were

again found to be the major cause of fault-injection in their SRS document. Most of the faults in both Team 4's and Team 5's SRS's were mapped back to Clerical Errors.

Furthermore, error abstraction using the HEAA tool can not only provide teams with insights about Error Types and Error Classes, but it can also provide insights about how prone their individual requirements engineering activities (i.e., elicitation, analysis, specification, management) are to different human error classes. To that end, the error proneness of requirements engineering activities to different human errors were analyzed (for each team separately). Table 31 provides the result of this analysis. It should be noted that for each Team's table, the sum total of all cells in the table is 100%.

For Team 1, most of the faults in their SRS were traced back to their elicitation and specification activities. Additionally, the major problem area for Team 1's elicitation activity was Lapses (Loss of Information errors and Accidentally Overlooking Requirements). Team 1's specification activity mainly suffered from Slips (Clerical errors and errors due to Lack of Consistency when writing specifications). As for Team 2, their most error-prone activity was the specification activity and it mainly suffered from Slips committed by the team members when writing the specifications. Most of the faults in Team 3's SRS documents were mapped back to elicitation activity and Team 3's elicitation activity mainly suffered from Lapse errors. As for Team 4, most of the faults in their SRS were traced back to specification and elicitation activities, with both activities suffering mainly from Clerical errors committed by team members.

Team 5's most error-prone activity was specification activity and a majority of human errors committed by Team 5's members were Clerical slips.

Table 31. Study 7: Proneness of Requirements Engineering Activities to Different Human Error Classes

Team 1 - DC	Slips		Lapses		Mistakes									
	Clerical	LC	LI	Accd.	Appln.	Env.	IM	WA	PU	MB	NH	LA	PS	IR	Synt.
Elicitation	1.7%		13.6%	25.4%	6.8%			1.7%	1.7%						
Analysis	6.8%				5.1%			3.4%							
Specification	15.3%	17.0%													
Management							1.7%								
Team 2 - SOSS	Slips		Lapses		Mistakes									
	Clerical	LC	LI	Accd.	Appln.	Env.	IM	WA	PU	MB	NH	LA	PS	IR	Synt.
Elicitation	2.9%			8.8%				8.8%				5.9%			
Analysis					5.9%										
Specification	52.9%	14.7%													
Management															
Team 3 - CMS	Slips		Lapses		Mistakes									
	Clerical	LC	LI	Accd.	Appln.	Env.	IM	WA	PU	MB	NH	LA	PS	IR	Synt.
Elicitation			15.4%	35.9%											
Analysis					12.8%										
Specification	23.08%														
Management							5.1%							7.7%	
Team 4 - WoW	Slips		Lapses		Mistakes									
	Clerical	LC	LI	Accd.	Appln.	Env.	IM	WA	PU	MB	NH	LA	PS	IR	Synt.
Elicitation	22.9%			8.3%	2.1%			4.2%							
Analysis	8.3%				4.2%	4.2%									2.1%
Specification	37.5%					4.2%									
Management														2.1%	
Team 5 - SBREB	Slips		Lapses		Mistakes									
	Clerical	LC	LI	Accd.	Appln.	Env.	IM	WA	PU	MB	NH	LA	PS	IR	Synt.
Elicitation											2.6%				
Analysis						15.8%									
Specification	42.1%	13.2%					18.4%							7.9%	
Management															

Overall, the analysis performed for RQ3 shows that, by abstracting human errors from their faults, software development teams can generate valuable insights about the major problem areas in their requirements engineering process. These insights can help development teams make future decisions about how to improve their requirements engineering process.

6.4. Summary of Results Obtained from Studies 4, 5, 6, and 7

Studies 4, 5, 6, and 7 validated the following:

- The usefulness of the human error-based requirements inspection approach (i.e., the Error Abstraction and Inspection or EAI approach) when it is supported by the newly developed Human Error Abstraction Assist or HEAA tool.
- The improvements in error abstraction accuracy provided by the HEAA tool during the error abstraction leg of the EAI inspection approach.

Overall, during Studies 4 and 5, it was found that the EAI approach, when added to the traditional fault-checklist inspections, can provide significant improvements in the defect detection effectiveness of inspectors (see Figure 19 and 20). Study 5 also revealed the strategies used by successful inspectors that helped them find additional faults during the error-informed reinspection step of the EAI approach. Inspectors will be trained on these reinspection strategies (shown in Table 25) in future evaluations of the EAI approach.

Additionally, results (in Tables 26 and 27) showed that the introduction of HEAA tool has helped increase the error abstraction accuracy of inspectors. An increased error abstraction accuracy means that inspectors are able to better understand the human error causes of requirements faults, which can help them find more faults related to the human errors during error-informed reinspection step of the EAI inspection approach.

Another major result obtained during Study 5 and Study 6 was that, when using the HEAA tool, inspectors faced difficulties when selecting the right requirements engineering activity wherein the fault originated. Requirements engineering professionals in Study 6 suggested that improving the training around requirements engineering activities can help alleviate this problem.

The four studies described in this chapter and the three studies described in Chapter 4 focused mainly on the usefulness of human error information for detecting errors and faults in requirements documents. The next chapter describes a study that involved industry professionals and focused on creating prevention strategies that can help requirements engineering teams avoid committing the human errors.

7. ERROR AND FAULT PREVENTION

Fault prevention can be described as the process of using the *knowledge of likely problems to prevent those problems from happening in future*. In software engineering research, the *knowledge of likely problems* is collected through historical fault/defect data, or expert opinion [50]–[52]. Fault prevention strategies that are based on historical fault data (i.e., a sample of faults) can provide specific measures/strategies to prevent those type of faults [50], [53]. On those lines, the human error information identified during the creation of the Human Error Taxonomy (Figure 7) presents an opportunity to create prevention strategies that can help in preventing the human errors from being committed by requirements engineers, and consequently the faults that are injected due to these human errors can be reduced as well.

To that end, an industrial survey was conducted at a software development organization (which is based out of Minneapolis, MN). Industry requirements professionals were trained on human error types (slip, lapse, mistakes) and the various human error classes and were asked to indicate the approaches (i.e., strategies) that they use in order to eliminate or reduce the likelihood for human errors from occurring.

Section 7.1 provide the research question and the study design, and Section 7.2 provide the result obtained from analyzing the survey data.

7.1. Study 8: Research Questions and Design

Table 32 provides the research question that was formulated during Study 8.

Table 32. Study 8: Research Question

#	Research Question
RQ1	What specific prevention strategies do industry practitioners employ for the human errors described in the Human Error Taxonomy?

The participants of Study 8 were 11 industry practitioners working in a software development organization based in Minneapolis, MN. The participants were first provided a video-based training on the human errors in the Human Error Taxonomy (HET). The training included: a module that helped participants understand the human errors types (slips, lapses, and mistakes), and a module that helped them understand the 15 human error classes in the Human Error Taxonomy. An additional module trained the participants on the Human Error Abstraction Assist or HEAA tool and how to use the tool to abstract human errors from requirements faults. This additional module was meant to provide participants with a deeper understanding about: how the human errors can lead to faults being injected in requirements documents. Post-training, participants were provided with a set of requirements faults, the complete information about each fault, a training supplement document containing description of each human error class, and an error form with their perception of the human error that caused fault-injection. Next, the participants were asked to answer the following survey item about each error:

How would you reduce the future occurrence of the human error? (Note that participants provided subjective feedback for this survey item).

7.2. Study 8: Data Analysis and Results

This section provides the results of analyzing the data gathered during Study 8. This section is organized around the research question that was provided in Table 32.

7.2.1. What Specific Prevention Strategies do Industry Practitioners Employ for the Human Errors Described in the Human Error Taxonomy?

The participants provided subjective feedback for the following survey item: *How would you reduce the future occurrence of the human error?* In their feedback, participants described the prevention mechanisms that they use in order to eliminate the occurrence of specific human

errors. This feedback, containing prevention mechanisms, was first analyzed separately for each human error class. If the description of the prevention mechanism was found to be incomplete or incomprehensible, the mechanism was rejected. Next, from the remaining mechanisms, those prevention mechanisms that were similar were grouped together. As the reported prevention strategies were being analyzed, it was observed that four high-level groups emerged based on the problem that was being addressed by the reported strategies/mechanisms:

- *Prevention mechanisms for communication problems*: Under this high-level category, participants described the prevention mechanisms for those human errors that result from cognitive under specification caused due to communication problems within the requirements engineering team and also the communication problems between the team and the end-users/stakeholders. Table 33 shows the prevention strategies for communication problems.
- *Prevention mechanisms through changes to resources*: Under this high-level category, participants described the prevention mechanisms for those human errors that result from unavailability of expert knowledge about the system-being-built. Table 34 shows the prevention mechanisms for this category.
- *Prevention mechanisms for management/administration problems*: Under this high-level category (see Table 35), participants reported strategies that can help prevent human errors through some administrative changes.
- *Prevention mechanisms through changes to requirements engineering (RE) procedure*: Under this high-level category (see Table 36), participants reported strategies that can help prevent human errors through changes to requirements engineering best practices.

Tables 33, 34, 35, and 36 provide prevention mechanisms that were obtained as a result of analyzing Study 8's data.

Table 33. Study 8: Prevention Mechanisms for Communication Problems

Prevention Strategy	Relevant Error Class in HET	Relevant Requirements Engineering Activity
Creating a communication plan that includes what type of communication should happen between the different team members, how should it happen (method of communication), and at what times (weekly, daily, or after completion of specific tasks, etc.).	Clerical Errors (Slip)	Specification, Analysis
Requirements team should get a list of common terminology from end-users and make sure all members are familiar with them. It is almost impossible to expect different end-users will use the same words/names for an entity.	Lack of Consistency in Requirements Specifications (Slip)	Specification
During requirements gathering, repeat back all the requirements that were heard and get confirmation that: (a) requirements were understood correctly (b) the end-users haven't missed any special circumstances	Accidentally Overlooking requirements (Lapse)	Elicitation
Creating a dictionary/glossary of terms used by clients. Notes gathered from different clients should be examined to check if different terms are being used for the same entity. If so, then the different terms should be consolidated into one name (in consultation with the clients).	Wrong Assumptions (Mistake)	Analysis
The requirements gathering person should ask the right follow-up questions in order to force the client to be more concise and clear in their use of terminology and to avoid redundant terms.	Wrong Assumptions (Mistake)	Elicitation
Knowledge transfer within the requirements engineering team should be encouraged. This can be done by asking requirements engineering team members to do presentations or talks about the parts of the system that they are currently working on. Additionally, end-users/stakeholders should be invited to such presentations in order to get feedback from them about: whether they think the requirements engineering team members' knowledge about the system is correct.	Poor/Low Understanding of Roles (Mistake)	Elicitation, Analysis
A glossary of important items/terms/entities related to the system-under-development should be created and distributed to all members of the requirements engineering team. It should also be continually updated.	Inadequate Requirements Process (Mistake)	Management

Table 34. Study 8: Prevention Mechanisms through Changes to Resources

Prevention Strategy	Relevant Error Class in HET	Relevant Requirements Engineering Activity
If the complete knowledge of system is not present, then application errors can be avoided by consulting Subject matter Experts (SMEs). Organizations have SMEs for the various parts of the system (being built) and they can provide functional knowledge about the system.	Application Errors (Mistake)	Analysis
Hiring an experienced *requirements gathering person* and training them on the dos and don'ts of requirements elicitation.	Wrong Assumptions (Mistake)	Elicitation
Subject matter Experts or SMEs, who can provide functional knowledge about the system and various parts of the system, should be available. Having technical SMEs on hand is also useful and they can help in choosing the right solutions related to design and programming constraints.	Problem-solution Errors (Mistake)	Analysis

Table 35. Study 8: Prevention Mechanisms for Management/Administration Problems

Prevention Strategy	Relevant Error Class in HET	Relevant Requirements Engineering Activity
From a very early stage in requirements engineering, a process should be in place that ensures that 'customer approvals' can be readily obtained by requirements engineering team members. Not having too many layers between requirements engineering team members and end-users can help with this.	Inadequate Requirements Process (Mistake)	Management
At the very outset of the requirements phase, the organization should establish a procedure for dealing with requirement-change. Procedures should be in place to ensure that team performs impact analysis for all proposed changes, and channels should be in place to get approvals from appropriate stakeholders.	Inadequate Requirements Process (Mistake)	Management

Table 36. Study 8: Prevention Mechanisms through Changes to RE Procedures

Prevention Strategy	Relevant Error Class in HET	Relevant Requirements Engineering Activity
Wherever applicable, when creating formal requirements specifications from elicited (and analyzed) requirements, the requirements author should make sure to get any formulas (mathematical expressions) validated from end-users/stakeholders who have supplied the formulas.	Clerical Errors (Slip)	Specification
Once the formal requirements specifications have been created, a workshop or JAD (Joint Application development) session should be conducted wherein end-users, programmers, testers, and system designers can review the document and discuss issues with the requirements engineering team.	Lack of Consistency in Requirements Specifications (Slip)	Specification
Building dependency and traceability matrices early in the requirements engineering phase and keeping them updated can help avoid overlooking any requirements. Members of requirements engineering team should be encouraged to utilize these matrices when eliciting/analyzing/writing requirements.	Accidentally Overlooking requirements (Lapse)	Elicitation
Both requirements engineering team and end-users should be encouraged to review the dependency matrix before requirements gathering sessions.	Accidentally Overlooking requirements (Lapse)	Elicitation
If the new system is being built is going to replace a legacy system, then it is essential that the requirements team gains as much knowledge as possible about the existing legacy system. But if the system is one of a kind, then techniques such as creating use case scenarios and showing them to the end-users/stakeholders can help gather knowledge about the system.	Application Errors (Mistake)	Analysis, Elicitation
When analyzing requirements, it is essential to think from a tester's perspective and validate every requirement being created. More specifically, requirements that describe formulas and mathematical expressions can easily be validated (by using techniques like boundary value analysis).	Application Errors (Mistake)	Analysis
A central reference guide for all variables used during requirements creation should be created and updated periodically. The guide should also define the relationships (i.e., how a change in one variable effects another) between the different variables. Requirements team should ensure that they should refer to the guide whenever they are eliciting/analyzing/writing requirements that use variables.	Information Management Errors (Mistake)	Management, Analysis, Specification
Creating data flow diagrams (DFD) using requirements specifications can reveal omissions. RE teams should be trained on DFD-construction and creating DFDs should be part of requirements engineering best practices.	Information Management Errors (Mistake)	Management
Inconsistencies can be avoided by limiting the number of places in the requirements document where the same item/entity (e.g., an equipment) is discussed. If this is not possible, then the requirements engineering team needs to maintain a log of related requirements within the documents.	Information Management Errors (Mistake)	Management, Specification
Assumptions are common issues, especially when a person thinks they know a lot about the system. Assumptions can be avoided by asking all members of requirements engineering team to create a central repository of any and all assumptions that they make while creating the requirements and making this repository visible to the entire team (so that team members can flag those assumptions that they think are incorrect).	Wrong Assumptions (Mistake)	Elicitation, Analysis
If the team does not have the proper know-how about what are the correct resources/techniques for creating the solution of a given problem, then it is a good idea to create a proof-of-concept (POC) first. The POC can reveal whether the solution that the team has come up with is the right one or not.	Problem-solution Errors (Mistake)	Analysis

Analyzing the data gathered during Study 8 has revealed a list of strategies that industrial practitioners use in order to avoid the occurrence of human errors described in the Human Error Taxonomy. It is anticipated that by applying these strategies, software development organizations can help the teams in avoiding the human errors and related faults, thereby increasing the quality of requirements created by the teams.

The next chapter provides a discussion of the major implications of the results obtained during the research described in this dissertation.

8. A DISCUSSION ON THE IMPLICATIONS OF RESULTS

This chapter provides a discussion of the implications of the various study-findings (i.e., results) obtained during the research described in this dissertation. Here, I discuss the results of answering the various research questions and the implications of these results. This chapter is organized around the several research questions (discussed in Chapter 3, 4, 6 and 7) that were driving the research described in this dissertation.

What types of requirements engineering human errors does the software engineering and psychology literature describe, and how can we organize the identified human errors identified into a taxonomy?

The systematic literature review (described in Chapter 3) identified the human errors that are most frequently committed during the requirements engineering phase of the software development life-cycle. The outcome of the systematic literature review was a Human Error Taxonomy (Figure 7) containing requirements phase human errors. The development of Human Error Taxonomy required close collaboration with a human error expert from psychology. While a number of general frameworks for classifying human errors have been proposed by psychology researchers, the errors found in software engineering literature did not utilize these frameworks. Therefore, each error description needed to be carefully analyzed in order to determine whether it truly represented a human error. Research described in this dissertation has shown that a close interaction between software engineers and psychology researchers can help in providing a theoretically-sound human error framework for organizing requirements engineering human errors. Furthermore, the human error taxonomy developed in this research will help software development teams identify the most frequently committed human errors so that they can focus the requirements inspection process on identification and removal of the faults caused by those

human errors, and also create and implement strategies to prevent the human errors (e.g. checklists, and trainings).

Does the human error-based inspection approach (i.e., the Error Abstraction and Inspection or EAI approach) improve the fault detection effectiveness of inspectors when compared to traditional requirements inspection approach?

Multiple empirical studies described in this dissertation provided evidence that the Error Abstraction and Inspection approach can improve the fault detection effectiveness of inspectors when compared to traditional inspections techniques. It should be noted that the Error Abstraction and Inspection approach works as an addendum to the traditional fault-checklist inspection approach. Results from Study 4 (Figure 19) showed that, on an average, the Error Abstraction and Inspection approach (supported by Human Error Abstraction Assist tool) increased the fault detection effectiveness of inspectors by 225% as compared to conducting only fault-checklist inspection on the requirements document. That is, during Study 4, a significantly large number of faults were identified by the inspectors during error-informed reinspection that were left undetected during the first inspection (the first inspection was a fault-checklist based inspection). Overall, it can be concluded that knowledge of human errors can aid software development teams in finding additional faults related to those human errors. The central idea of the Error Abstraction and Inspection (EAI) approach is that once a development team becomes aware of the human errors that were committed during an SRS's development process, it is likely that faults related to those human error are also present in the SRS. Furthermore, analyzing the data collected during Study 7 (see Table 25) helped in uncovering the strategies that successful inspectors use during the error-informed reinspection leg of the EAI inspection approach. It is

anticipated that by training inspectors on these reinspection strategies can further improve the fault detection effectiveness of the EAI approach.

Does the Human Error Abstraction Assist tool provide a useful method for abstracting human errors from requirements faults?

During the initial evaluations, the Error Abstraction and Inspection (EAI) approach was supported by the Human Error Taxonomy (HET). Then, the Human Error Abstraction Assist or HEAA tool was developed (with HET as its foundation) and it was found that the HEAA tool helped inspectors in understanding the error abstraction process better. We found that implementing the steps of a standard error abstraction process (in HEAA tool) helped the inspectors achieve better error abstraction performance. Furthermore, using feedback from participants, some improvements were made to the HEAA tool and the refined HEAA was able to further improve the error abstraction accuracy of inspectors. Results from Study 7, however, revealed that the participants face difficulties when applying HEAA tool to abstract errors from faults in externally-developed requirements documents (i.e., SRS documents). Therefore, there is a need for improving the error abstraction process and the HEAA tool when employing it on externally-developed SRS documents. Another major area wherein HEAA tool needs to be improved is helping inspectors select the correct requirements engineering activity where the fault (being analyzed for human error) originated. When using the HEAA tool, participants made most of the misjudgments when selecting the requirements engineering activity wherein the fault originated (i.e., the human error occurred and caused the injection of the fault being analyzed). Requirements engineering professional during a Live Conference study (Study 6) provided feedback that inspectors might have different understanding of requirements engineering activities (depending on the software development organization an inspector belongs to). This is

because organizations follow different requirements engineering processes and techniques. Therefore, requirements engineering professionals suggested that the error abstraction training should provide detailed descriptions of the various activities, so that all inspectors can develop a similar understanding of the activities before they start using the HEAA tool.

What specific prevention strategies do industry practitioners employ for the human errors described in the Human Error Taxonomy?

Study 8 compiled a list of the strategies or mechanisms that can be used by requirements engineering teams to reduce the incidence of human errors. A total of twenty-four (24) human error prevention strategies were identified during the survey. Of the 24 error prevention strategies, a majority of the strategies (17 of them) address *Mistake* error type in the Human Error Taxonomy. This showed that participants believed that *Mistake errors* represent a more deep-seated problem in the requirements engineering process. These problems are also called *latent errors* and require system-wide improvements/changes in order to reduce their occurrence. Additionally, strategies related to changes in requirements engineering procedures (Table 36) were the most frequently reported strategies. Almost half of all the identified strategies (i.e., 12 out of 24) are related to human errors that can be prevented via changes in requirements engineering practices. Participants emphasized that requirements engineering team can prevent many errors and faults by creating and maintaining the following requirements engineering tools: dependency matrix, traceability matrix, Data Flow Diagrams, and a central data dictionary.

9. CONCLUSION

This section discusses the major contribution of the work described in this dissertation to Software Engineering research and practice. This section also enlists the publications that will be the output of this dissertation work.

9.1. Contribution to Software Engineering Research and Practice

This research has illustrated that human error research has the potential to provide an effective solution to the software quality problem. Through a meticulous application of human error research to requirements engineering, this research has resulted in the development of a Human Error Taxonomy (HET) that is strongly grounded in human error theories.

This research empirically validated the usefulness of the HET to support a formal requirements inspection technique (the Error Abstraction and Inspection - EAI) that can be used by researchers and practitioners when understanding requirement errors at their organizations. This will also motivate other researchers to employ human error research for developing similar human error based quality improvement approaches for other software lifecycle phases. Furthermore, this research highlighted the need for a more formal Human Error Abstraction Assist (HEAA) tool to help software engineers systematically investigate the human error causes of requirements faults. Interested researchers might develop similar tools to understand the human error causes of problems that occur during other phases of software development. The systematic literature review (SLR) procedure that identified requirements phase during this research can be used by other interested researchers as a blueprint to identify human errors that happen in other phases of software development lifecycle.

The results from this work provide insight into the human error causes of defects and failures that occur during software development. These insights can be used by organizations and

developers to focus their review process on detection and removal of defects and to implement policies and interventions to prevent the most frequently occurring human errors.

This research has also compiled a preliminary list of the mechanisms that can be used by organizations to prevent the incidence of human errors during software development process. Reducing the incidence of human errors will lead to a reduction in the number of faults/defects and failures that are caused by the human errors, thereby increasing the overall quality of the software being developed.

9.2. Publications

This section describes the publications resulted from the work done for this dissertation. The publication plan is described in terms of articles (conference papers and journal papers) that have been published, and the articles that have been submitted or are in progress.

9.2.1. Refereed Conferences

1. Anu, V., Walia, G., Hu, W., Carver, J., and Bradshaw, G. “Issues and Opportunities for Human Error-based Requirements Inspections: An Exploratory Study”, *ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM 2017)* [54].
2. Anu, V., Walia, G., and Bradshaw, G. "Incorporating Human Error Education into Software Engineering Courses via Error-based Inspections", *48th ACM Technical Symposium on Computer Science Education (SIGCSE 2017)* [55].
3. Anu, V., Walia, G., Hu, W., Carver, J., and Bradshaw, G. “Using a Cognitive Psychology Perspective on Errors to Improve Requirements Quality: An Empirical Investigation” *Proceedings of 27th IEEE International Symposium on Software Reliability Engineering (ISSRE 2016)*[[56].

4. Anu, V., Walia, G., Hu, W., Carver, J., and Bradshaw, G. "Error Abstraction Accuracy and Fixation during Error-based Requirements Inspections" *Proceedings of 27th IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW 2016)* [57]
5. Hu, W., Carver, J., Anu, V., Walia, G., and Bradshaw, G. "Detection of Requirement Errors and Faults via a Human Error Taxonomy: A Feasibility Study" *Proceedings of 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM 2016)* [58]
6. Anu, V., Walia, G., Hu, W., Carver, J., and Bradshaw, G. "Effectiveness of Human Error Taxonomy during Requirements Inspection: An Empirical Investigation" *Proceedings of 28th International Conference on Software Engineering and Knowledge Engineering (SEKE 2016)* [59]

9.2.2. Refereed Journal Articles (Under Review and In progress)

1. Anu, V., Hu, W., Carver, J., Walia, G., and Bradshaw, G. "Development of a Human Error Taxonomy for Software Requirements: A Systematic Literature Review" Accepted with some changes to *Journal of Information and Software Technology (JI&ST), 2018*.

   This article describes the systematic literature review process for developing the Human Error taxonomy (HET). This paper has been modified based on reviewers' comments and re-submitted to JI&ST.
2. Hu, W., Carver, J., Anu, V., Walia, G., and Bradshaw, G. "Using Human Error Information for Error Prevention" Accepted to be published in *Empirical Software Engineering (EMSE), May 2018.*

3. Anu, V., Walia, G., Hu, W., Carver, J., and Bradshaw, G. “Progressive Refinement of a Human Error Detection Tool for Improving the Investigation of Human Error Causes of Requirements Faults.” In progress to be submitted to *Journal of Information and Software Technology (JI&ST), 2018*.
   This paper will describe the series of three controlled empirical studies that resulted in the development and refinement of a human error detection tool called Human Error Abstraction Assist (HEAA). The error abstraction data obtained during the empirical studies will be analyzed for more insights.

9.2.3. Workshops and Live Studies

1. Anu, V., Walia, G., Bradshaw, G., Hu, W., and Carver, J. "Using Human Error Abstraction Method for Detecting and Classifying Requirements Errors: A Live Study" *In 23rd International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ 2017)* [48]
2. Hu, W., Carver, J., Anu, V., Walia, G., and Bradshaw, G. "Understanding Human Errors in Software Requirements: An Online Survey", In *23rd International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ 2017)* [60]
3. First Workshop on Applications of Human Error to Improve Software Engineering. *Held in the International Conference on Software Engineering (ICSE 2015)*

9.3. Future Work

An immediate task is to replicate the empirical studies (which were conducted in academic settings) with professional developers in industrial settings. This will help in

understanding if the results obtained from students are consistent with those obtained from industry practitioners. Another future task is to extend the Human Error Taxonomy (HET) by collecting and analyzing error data from professional developers. HET currently contains requirements phase human errors that were found in software engineering literature. In order to add more error classes to HET, I plan to conduct *ethnographical studies*, wherein a participant observer (a human error expert) will act as a fly on the wall and take notes as professional developers carry out the various requirements engineering activities.

Another future goal is to improve the human error investigation tool (i.e., HEAA tool) by adding the human factors perspective of latent organizational errors. This will require reviewing the software engineering literature to identify the *organizational weaknesses* (like lack of time and resources allocated to requirements phase). These *organizational weaknesses* act as pre-cursors to the human errors. This kind of comprehensive human error and human factor investigation can provide organizations an opportunity to perform fine-grained analysis of the *people and process* problems that exist within their requirements engineering practices.

An area of future work is to develop and validate error taxonomies for the design and implementation phases of software development. Work has already begun on a research project that uses the systematic literature review process to identify the human errors that are committed during the architecture/design phase of the software development lifecycle.

Another future goal is to create and evaluate educational materials and procedures that can be used by academic educators and project managers to impart knowledge about human errors that affect the software development process. This research will benefit students and practitioners by providing insights into the human cognition aspect of software development. A detailed understanding of the psychological and cognitive processes that lead to human errors

will provide software engineers with a fresh perspective on software quality assurance and equip them with new set of tools to prevent, detect, and fix software faults.

Another future goal is to investigate other areas of *Software Engineering* and *Information Sciences* that can benefit from inclusion of Cognitive Psychology research on human errors and human factors. One research area that is of particular interest is: *incident investigation of successful cybersecurity attacks from a human factors analysis perspective*. This research will look at: what are the major cognitive and human factors that drive the erroneous behavior of the people involved in cybersecurity incidents.

REFERENCES

[1] M. T. Baysari, C. Caponecchia, A. S. McIntosh, and J. R. Wilson, "Classification of errors contributing to rail incidents and accidents: A comparison of two human error identification techniques," *Saf. Sci.*, vol. 47, no. 7, pp. 948–957, 2009.

[2] T. Diller, G. Helmrich, S. Dunning, S. Cox, A. Buchanan, and S. Shappell, "The Human Factors Analysis Classification System (HFACS) applied to health care.," *Am. J. Med. Qual.*, vol. 29, no. 3, pp. 181–90, 2014.

[3] T. B. Sheridan, "Understanding human error and aiding human diagnostic behaviour in nuclear power plants," in *Human detection and diagnosis of system failures*, 1981, pp. 19–35.

[4] D. Wiegmann and C. Detwiler, "Human Error and General Aviation Accidents : A Comprehensive , Fine-Grained Analysis Using HFACS," *Security*, no. December, pp. 1–5, 2005.

[5] B. Boehm and V. R. Basili, "Software Defect Reduction Top 10," *Computer (Long. Beach. Calif).*, vol. 34, no. 1, pp. 135–137, 2001.

[6] M. Hamill and K. Goseva-Popstojanova, "Common trends in software fault and failure data," *IEEE Trans. Softw. Eng.*, vol. 35, no. 4, pp. 484–496, 2009.

[7] J. C. Chen and S. J. Huang, "An empirical analysis of the impact of software development problem factors on software maintainability," *J. Syst. Softw.*, vol. 82, no. 6, pp. 981–992, 2009.

[8] R. Dion, "Process Improvement and the Corporate Balance Sheet," *IEEE Softw.*, vol. 10, no. 4, pp. 28–35, 1993.

[9] S. T. Knox, “Modeling the cost of software quality,” *Digit. Tech. J.*, vol. 5, pp. 9–16, 1993.

[10] D. Leffingwell, “Calculating the return on investment from more effective requirements management,” *Cut. IT J.*, vol. 10, no. 4, pp. 13–16, 1997.

[11] B. Brykczynski, “A survey of software inspection checklists,” *ACM SIGSOFT Softw. Eng. Notes*, vol. 24, no. 1, p. 82, 1999.

[12] D. L. Parnas and M. Lawford, “The role of inspection in software quality assurance,” in *IEEE Transactions on Software Engineering*, 2003, vol. 29, no. 8, pp. 674–676.

[13] A. A. Porter and L. G. Votta, “An experiment to assess different defect detection methods for software requirements inspections,” *Proc. 16th Int. Conf. Softw. Eng.*, pp. 103–112, 1994.

[14] T. Thelin, P. Runeson, and C. Wohlin, “An experimental comparison of usage-based and checklist-based reading,” in *IEEE Transactions on Software Engineering*, 2003, vol. 29, no. 8, pp. 687–704.

[15] E. Kantorowitz, A. Guttman, and L. Arzi, “The performance of the N-fold requirement inspection method,” *Requir. Eng.*, vol. 2, no. 3, pp. 152–164, 1997.

[16] J. Martin and W. T. Tsai, “N-Fold inspection: a requirements analysis technique,” *Commun. ACM*, vol. 33, no. 2, pp. 225–232, 1990.

[17] G. M. Schneider, J. Martin, and W. T. Tsai, “An experimental study of fault detection in user requirements documents,” *ACM Trans. Softw. Eng. Methodol.*, vol. 1, no. 2, p. 188, 1992.

[18] V. R. Basili, S. Green, O. Laitenberger, F. Shull, S. Sørumgård, and M. V. Zelkowitz, "The Empirical Investigation of Perspective-based Reading," *Empir. Softw. Eng.*, vol. 1, pp. 133–164, 1996.

[19] J. C. Maldonado, J. Carver, F. Shull, S. Fabbri, E. Dória, L. Martimiano, M. Mendonça, and V. Basili, "Perspective-based reading: A replicated experiment focused on individual reviewer effectiveness," in *Empirical Software Engineering*, 2006, vol. 11, no. 1, pp. 119–142.

[20] F. Shull, I. Rus, and V. Basili, "How perspective-based reading can improve requirements inspections," *Computer (Long. Beach. Calif).*, vol. 33, no. 7, pp. 73–79, 2000.

[21] A. A. Porter, V. R. Basili, and L. G. Votta, "Comparing Detection Methods for Software Requirements Inspections: A Replicated Experiment," *IEEE Trans. Softw. Eng.*, vol. 21, no. 6, pp. 563–575, 1995.

[22] O. Laitenberger, "A Survey on Software Inspection Technologies," in *Handbook on Software Engineering and Knowledge Engineering*, vol. 2, 2002, pp. 517–555.

[23] G. S. Walia and J. C. Carver, "A systematic literature review to identify and classify software requirement errors," *Inf. Softw. Technol.*, vol. 51, no. 7, pp. 1087–1109, 2009.

[24] J. W. Lee, Y. H. Lee, T. Il Jang, D. H. Kim, and J. Park, "A proposition of human factors approaches to reduce human errors in nuclear power plants," in *IEEE Conference on Human Factors and Power Plants*, 2007, pp. 16–22.

[25] I. A. Taib, A. S. McIntosh, C. Caponecchia, and M. T. Baysari, "A review of medical error taxonomies: A human factors perspective," *Safety Science*, vol. 49, no. 5. pp. 607–615, 2011.

[26] G. da S. R. Ribeiro, R. C. da Silva, M. de A. Ferreira, and G. R. da Silva, “Slips, lapses and mistakes in the use of equipment by nurses in an intensive care unit,” *Rev. da Esc. Enferm.*, vol. 50, no. 3, pp. 419–426, 2016.

[27] S. A. Shappell and D. A. Wiegmann, “The Human Factors Analysis and Classification System – HFACS,” *Security*, p. 19, 2000.

[28] N. A. Stanton and P. M. Salmon, “Human error taxonomies applied to driving: A generic driver error taxonomy and its implications for intelligent transport systems,” *Saf. Sci.*, vol. 47, no. 2, pp. 227–237, 2009.

[29] G. S. Walia and J. C. Carver, “Using error abstraction and classification to improve requirement quality: Conclusions from a family of four empirical studies,” *Empir. Softw. Eng.*, vol. 18, no. 4, pp. 625–658, 2013.

[30] REQB, “Standard glossary of terms used in Requirements Engineering,” *Requir. Eng. Qualif. Board*, vol. 1.0, p. 24, 2011.

[31] IEEE, “Systems and software engineering -- Vocabulary,” *ISO/IEC/IEEE 24765:2010(E)*, pp. 1–418, 2010.

[32] IEEE std, “IEEE Standard Glossary of Software Engineering Terminology (IEEE Std 610.12-1990). Los Alamitos,” *CA IEEE Comput. Soc.*, vol. 610.12-199, 1990.

[33] A. A. Alshazly, A. M. Elfatatry, and M. S. Abougabal, “Detecting defects in software requirements specification,” *Alexandria Eng. J.*, vol. 53, no. 3, pp. 513–527, 2014.

[34] F. Lanubile, F. Shull, and V. R. Basili, “Experimenting with Error Abstraction in Requirements Documents,” in *In Proceedings of the 5th International symposium on software metrics*, 1998.

[35] G. S. Walia, J. C. Carver, and P. Thomas, “Requirement Error Abstraction and Classification: An Empirical Study,” in *In Proceedings of the 5th International Symposium on Empirical Software Engineering*, 2006, pp. 336–345.

[36] G. S. Walia and J. C. Carver, “Evaluating the use of requirement error abstraction and classification method for preventing errors during artifact creation: A feasibility study,” in *In Proceedings of the 24th IEEE International Symposium on Software Reliability Engineering, ISSRE*, 2010, pp. 81–90.

[37] G. S. Walia, J. C. Carver, and T. Philip, “Requirement error abstraction and classification: A control group replicated study,” in *In Proceedings of the 18th IEEE International Symposium on Software Reliability Engineering, ISSRE*, 2007, pp. 71–80.

[38] R. P. E. Gordon, “The contribution of human factors to accidents in the offshore oil industry,” *Reliab. Eng. Syst. Saf.*, vol. 61, no. 1, pp. 95–108, 1998.

[39] J. Reason, *Human error*. New York, NY: Cambridge University Press, 1990.

[40] J. Rasmussen and K. J. Vicente, “Coping with human errors through system design: implications for ecological interface design,” *Int. J. Man. Mach. Stud.*, vol. 31, no. 5, pp. 517–534, 1989.

[41] J. Rasmussen, “Skills Rules and Knowledge, Other Distinctions in Human Performance Models,” *IEEE Trans. Syst. Man. Cybern.*, vol. 13, no. 3, pp. 257–266, 1983.

[42] J. Rasmussen, “Human errors. A taxonomy for describing human malfunction in industrial installations,” *J. Occup. Accid.*, vol. 4, no. 2–4, pp. 311–333, 1982.

[43] J. Rasmussen, “Skills, Rules, and Knowledge; Signals, Signs, and Symbols, and Other Distinctions in Human Performance Models,” *IEEE Trans. Syst. Man Cybern.*, vol. SMC-13, no. 3, pp. 257–266, 1983.

[44] D. A. Norman, *The Design of Everyday Things*, vol. 16, no. 4. 2002.

[45] P. Brereton, B. A. Kitchenham, D. Budgen, M. Turner, and M. Khalil, "Lessons from applying the systematic literature review process within the software engineering domain," *J. Syst. Softw.*, vol. 80, no. 4, pp. 571–583, 2007.

[46] D. Budgen and P. Brereton, "Performing systematic literature reviews in software engineering," in *Proceeding of the 28th international conference on Software engineering - ICSE '06*, 2006, p. 1051.

[47] B. Kitchenham, "Procedures for performing systematic reviews," *Keele, UK, Keele Univ.*, vol. 33, no. TR/SE-0401, p. 28, 2004.

[48] V. Anu, G. S. Walia, W. Hu, J. C. Carver, and G. Bradshaw, "Using Human Error Abstraction Method for Detecting and Classifying Requirements Errors: A Live Study," in *23rd International Working Conference on Requirements Engineering: Foundation for Software Quality*, 2017.

[49] A. Esgate, D. Groome, and K. Baker, *An Introduction to Applied Cognitive Psychology*. Psychology Press, 2005.

[50] G. M., "Software defect prevention using orthogonal defect prevention.," 2005. .

[51] S. Kumaresh and R. Baskaran, "Software Defect Prevention through Orthogonal Defect Classification," *Int. J. Comput. Technol.*, vol. 11, no. 3, pp. 2393–2400, 2014.

[52] S. Kumaresh and R. Baskaran, "Defect Analysis and Prevention for Software Process Quality Improvement," *Int. J. Comput. Appl.*, vol. 8, no. 7, pp. 42–47, 2010.

[53] W. Hu, J. C. Carver, V. Anu, G. S. Walia, and G. Bradshaw, "Using human error information for error prevention," *Empir. Softw. Eng.*, 2018.

[54] V. Anu, G. Walia, W. Hu, J. C. Carver, and G. Bradshaw, “Issues and Opportunities for Human Error-Based Requirements Inspections: An Exploratory Study,” in *International Symposium on Empirical Software Engineering and Measurement*, 2017, vol. 2017–November, pp. 460–465.

[55] V. Anu, G. Walia, and G. Bradshaw, “Incorporating Human Error Education into Software Engineering Courses via Error-based Inspections,” in *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education - SIGCSE ’17*, 2017, pp. 39–44.

[56] V. Anu, G. S. Walia, W. Hu, J. C. Carver, and G. Bradshaw, “Using A Cognitive Psychology Perspective on Errors to Improve Requirements Quality: An Empirical Investigation,” in *IEEE 27th International Symposium on Software Reliability Engineering (ISSRE’16)*, 2016, pp. 65–76.

[57] V. Anu, G. S. Walia, W. Hu, J. C. Carver, and G. Bradshaw, “Error Abstraction Accuracy and Fixation during Error-based Requirements Inspection,” in *27th IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW ’16).*, 2016.

[58] W. Hu, J. C. Carver, V. Anu, G. S. Walia, and G. Bradshaw, “Detection of Requirement Errors and Faults via a Human Error Taxonomy: A Feasibility Study,” in *10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM*, 2016.

[59] V. Anu, G. S. Walia, W. Hu, J. C. Carver, and G. Bradshaw, “Effectiveness of Human Error Taxonomy during Requirements Inspection: An Empirical Investigation,” in *2016 International Conference on Software Engineering and Knowledge Engineering, SEKE*, 2016.

[60] W. Hu, J. C. Carver, V. Anu, G. S. Walia, and G. Bradshaw, “Understanding Human Errors in Software Requirements: An Online Survey,” in *23rd International Working Conference on Requirements Engineering: Foundation for Software Quality*, 2017.

APPENDIX A. PAPERS THAT PROVIDED INPUT TO THE HET DURING THE SLR PROCESS

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.

[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 M V., 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 B. HUMAN ERROR ABSTRACTION ASSIST (HEAA) – INITIAL VERSION

1. Choose one of the following options to decide where the fault originated:

(a) Did the fault occur

- While the system was being analyzed?
- While a large system was being divided into smaller parts?
- While system functionalities (functional requirements) and system behavior (performance and other non-functional requirements) were being determined?

(b) Did the fault occur during interviews or discussions with the stakeholders (end users, project sponsors, etc.)? This is where the user needs are gathered.

(c) Did the fault occur when the system information/requirements were being documented to create a formal software requirements document?

(d) Did the fault occur

- During the *management* of the activities in a), b), or c) above?
- As requirements evolved or changed (i.e., traceability, version control, etc.)

RE activity associated to each option:

Option (a) – Requirement Analysis. **Option (b) –** Requirement Elicitation.
Option (c) – Requirement Specification. **Option (d) –** Requirement Management

2. Please consider the task which was being performed when the fault was injected (i.e., when the human error occurred) and form a task/problem statement. For example,

"Analyzing the number of available parking spaces in the parking garage to arrive at a generic formula for calculating the number of available parking spaces (a = k-r)."

Note that you will be asked to provide this task/problem statement in the 'Error-Report Form'

The boxes below provide the human errors that are relevant to various RE activities. Based on your answer to Question# 1, go to the appropriate box and pick the human error you think caused the fault.

Refer to the **HET details document to get a detailed description of the human error and an example fault.

Note that you will be asked to provide a brief description of why you picked a specific human error in the 'Error-Report Form'.

Requirement Analysis

- **Application error:** requirement analyst's misunderstanding or lack of knowledge of a part of (or the whole) system or problem
- **Environment error:** misunderstanding or misuse of the requirement analysis tools available for use in the project
- **Wrong assumptions** made by requirement analyst about user/stakeholder needs or opinions or any incorrect assumptions by RE analysts
- **Low understanding of each other's roles**: RE analyst does not understand the roles of all end users, stakeholders and other RE analysts.
- **Mistaken belief** of RE analysts that it is impossible to specify non-functional requirements in a verifiable form
- **Problem-Solution errors:** Lack of knowledge of the requirement analysis process and general requirement engineering know-how

Requirement Elicitation

- **Clerical Error:** Carelessness while recording user needs
- **Loss of information from stakeholders:** Forgetting, discarding or failing to store information or documents provided by stakeholders.
- **Accidentally overlooking requirements**
- **Application error:** stakeholder's or requirement gathering person's misunderstanding of a part of (or the whole) system or problem
- **Environment error:** misunderstanding or misuse of the requirement gathering tools available for use in the project
- **Wrong assumptions** made by requirement gathering person about user/stakeholder needs or opinions or any incorrect assumptions made by requirement gathering person.
- **Low understanding of each other's roles:** Requirement gathering person does not understand the roles of all end users and stakeholders.
- **Mistaken belief** of requirement gathering person that it is impossible to specify non-functional requirements in a verifiable form
- **Not having a clear demarcation between client and users**: Requirement gathering person's misunderstanding of the difference between clients and users
- **Lack of awareness of sources of requirements**

Requirement Specification

- **Clerical Error:** Carelessness while documenting specifications from elicited requirements.
- **Lack of consistency In Requirement Specifications:** Lack of logical coherence in the requirement specification documentation, which makes it difficult to be interpreted correctly
- **Environment error:** misunderstanding or misuse of the requirement specification tools available for use in the project
- **Syntactic error:** Misunderstanding of grammatical rules of natural language (English) or grammatical rules of a formal requirement specification language.

Requirement Management

- **Inadequate Requirements Process:** All steps required to ensure a robust requirement engineering process are not followed
- **Information Management error**: lack of knowledge about standard procedures and practices defined by the organization

APPENDIX C. REFINED HUMAN ERROR ABSTRACTION ASSIST (HEAA) TOOL

Step 1:

Choose one of the following options to decide where the fault originated (i.e., where the human error occurred and caused the fault to be inserted):

(a) Did the fault occur:

- While the system was being analyzed?
- While a large system was being divided into smaller parts?
- While system functionalities (functional requirements) and system behavior (performance and other non-functional requirements) were being determined?

(b) Did the fault occur during interviews or discussions with the stakeholders (end users, project sponsors, etc.)? This is where the user needs are gathered.

(c) Did the fault occur when the system information/requirements were being documented to create a formal software requirement document?

(d) Did the fault occur:

- During the *management* of the activities in a), b), or c) above?
- As requirements evolved or changed (i.e., traceability, version control, etc.)

RE activity associated to each option:

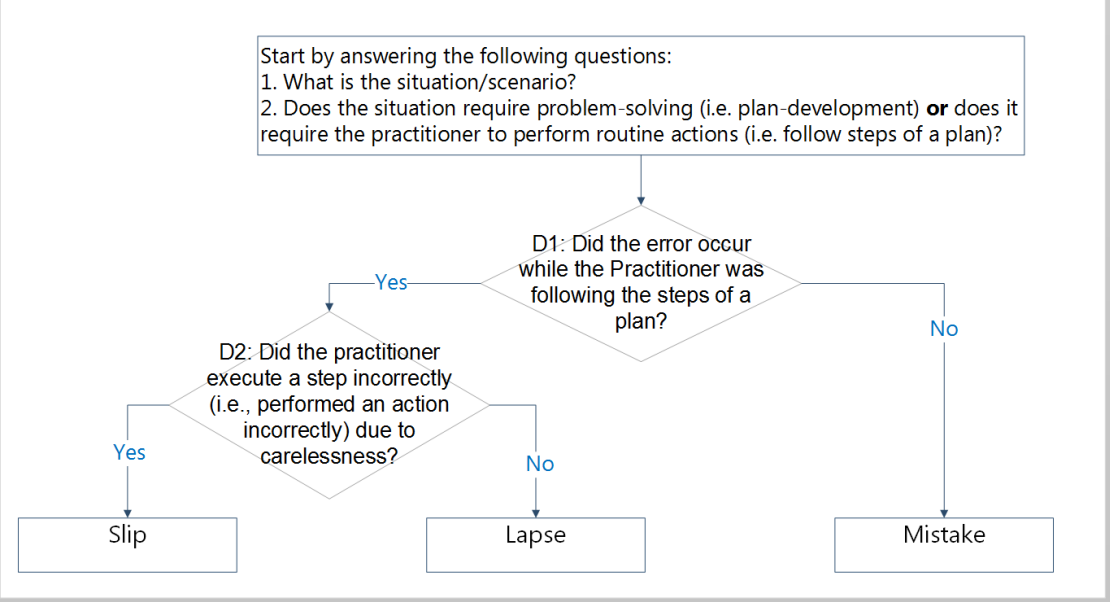
Option (a) – Requirement Analysis. **Option (b)** – Requirement Elicitation.

Option (c) – Requirement Specification. **Option (d)** – Requirement Management

Step 2:

Please consider the *task*, which was being performed, and form a task/scenario statement. In order to do this, try to visualize the scenario where you think the human error might have occurred.

Based on the scenario, *use the decision tree below* to decide whether the human error was a slip, a lapse, or a mistake



The boxes on the next two pages provide the human errors that are relevant to various RE activities. Based on your RE activity choice in Step 1 **and** human error type choice during Step 2 (slip, lapse, or mistake), go to the appropriate box and pick the human error you think caused the fault.

You can refer to the HET details document to get a detailed description of the human error and an example fault.

Step 3: Pick the appropriate Human Error

Requirement Analysis

Slips:

- **Clerical Error:** Carelessness while analyzing elicited requirements

Mistakes:

- **Application error:** analyst's misunderstanding or lack of knowledge of a part of (or the whole) system or problem
- **Environment error:** misunderstanding or misuse of the requirement analysis tools available for use in the project
- **Wrong assumptions** made by requirement analyst about user/stakeholder needs or opinions or any incorrect assumptions by RE analysts
- **Low understanding of each other's roles**: RE analyst does not understand the roles of all end users, stakeholders and other RE analysts.
- **Mistaken belief** of RE analysts that it is impossible to specify non-functional requirements in a verifiable form
- **Problem-Solution errors:** Lack of knowledge of the requirement analysis process and general requirement engineering know-how

Requirement Elicitation

Slips:

- **Clerical Error:** Carelessness while recording user needs

Lapses:

- **Loss of information from stakeholders:** Forgetting, discarding or failing to store information or documents provided by stakeholders.
- **Accidentally overlooking requirements:** Overlooking a requirement or some information that is crucial to the requirement

Mistakes:

- **Application error:** stakeholder's or requirement gathering person's misunderstanding of a part of (or the whole) system or problem
- **Environment error:** misunderstanding or misuse of the requirement gathering tools available for use in the project
- **Wrong assumptions** made by requirement gathering person about user/stakeholder needs or opinions or any incorrect assumptions made by requirement gathering person.
- **Low understanding of each other's roles:** Requirement gathering person does not understand the roles of all end users and stakeholders.
- **Mistaken belief** of requirement gathering person that it is impossible to specify non-functional requirements in a verifiable form
- **Not having a clear demarcation between client and users**: Requirement gathering person's misunderstanding of the difference between clients and users
- **Lack of awareness of sources of requirements**

Requirement Specification

Slips:

- **Clerical Error:** Carelessness while documenting specifications from elicited requirements.
- **Lack of consistency In Requirement Specifications:** Lack of logical coherence in the requirement specification documentation, which makes it difficult to be interpreted correctly

Mistakes:

- **Environment error:** misunderstanding or misuse of the requirement specification tools available for use in the project
- **Syntactic error:** Misunderstanding of grammatical rules of natural language (English) or grammatical rules of a formal requirement specification language.

Requirement Management

Mistakes:

- **Inadequate Requirements Process:** All steps required to ensure a robust requirement engineering process are not followed
- **Information Management error**: lack of knowledge about standard procedures and practices defined by the organization

APPENDIX D. STUDY 7 - TEAMS AND SYSTEM DESCRIPTIONS

Team #	# of members	System Name and Description (Length of SRS Document)
1	8	Dissertation Calculator (DC): The DC application will allow graduate students to easily create a calendar with specific events and deadlines related to their dissertation progress. (7-page long SRS)
2	6	Science Olympiad Scoring System (SOSS): Function of SOSS is to store data for Science Olympiad competitions at NDSU. The goal of the system is to provide a central location for judges to view and edit scores for each event of the competition. (9-page long SRS)
3	7	Capstone Management System (CMS): Currently, Excel Spreadsheets and handwritten notes are used to administratively manage the computer science Capstone class and projects. The CMS project will develop of a set of tools for management of the Capstone Class. It will include the ability to authenticate users of Student and Admin type, project–bidding, profile view/edit, etc. (8-page long SRS)
4	7	Wonders of Weather (WoW): The WoW system will allow a course instructor to create a class in which the students can enter weather data on specified days. Instructor chooses which data is required on specific days and can enter the data on the required day. The system will also keep track of the students' grades. (5-page long SRS)
5	8	Sugar Beet Research and Education Board (SBREB): Currently, The Sugar Beet Research and Education Board in North Dakota uses a physical paper medium for collecting and storing their grant proposals. SBREB will be an online interface for submitting, storing, and reviewing grant proposals and associated research. (6-page long SRS)