Information and Software Technology 51 (2009) 1087–1109

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

A systematic literature review to identify and classify software requirement errors

Gursimran Singh Walia [a], Jeffrey C. Carver [b,*]

[a] Mississippi State University, Department of Computer Science and Engineering, 300 Butler Hall, Mississippi State, MS 39762, United States
[b] University of Alabama, Computer Science, Box 870290, 101 Houser Hall, Tuscaloosa, AL 35487-0290, United States

ARTICLE INFO

Article history:
Received 7 December 2007
Received in revised form 27 January 2009
Accepted 29 January 2009
Available online 5 March 2009

Keywords:
Systematic literature review
Human errors
Software quality

ABSTRACT

Most software quality research has focused on identifying faults (i.e., information is incorrectly recorded in an artifact). Because software still exhibits incorrect behavior, a different approach is needed. This paper presents a systematic literature review to develop taxonomy of errors (i.e., the sources of faults) that may occur during the requirements phase of software lifecycle. This taxonomy is designed to aid developers during the requirement inspection process and to improve overall software quality. The review identified 149 papers from the software engineering, psychology and human cognition literature that provide information about the sources of requirements faults. A major result of this paper is a categorization of the sources of faults into a formal taxonomy that provides a starting point for future research into error-based approaches to improving software quality.

Contents

* Corresponding author. Tel.: +1 205 348 9829; fax: +1 205 348 0219.
E-mail addresses: gw86@cse.msstate.edu (G.S. Walia), carver@cs.ua.edu (J.C. Carver).

0950-5849/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.infsof.2009.01.004

1. Introduction

The software development process involves the translation of information from one form to another (i.e., from customer needs to requirements to architecture to design to code). Because this process is human-based, mistakes are likely to occur during the translation steps. To ensure high-quality software (i.e., software with few faults), mechanisms are needed to first prevent these mistakes and then to identify them when they do occur. Successful software organizations focus attention on software quality, especially during the early phases of the development process. By identifying problems early, organizations reduce the likelihood that they will propagate to subsequent phases. In addition, finding and fixing problems earlier rather than later is easier, less expensive, and reduces avoidable rework [19,28].

The discussion of software quality focuses around the use of a few important terms: *error*, *fault*, and *failure*. Unfortunately, some of these terms have competing, and often contradictory, definitions in the literature. To alleviate confusion, we begin by providing a definition for each term that will be used throughout the remainder of this paper. These definitions are quoted from Lanubile et al. [53], and are consistent with software engineering textbooks [32,67,77] and an IEEE Standard [1].

> *Error – defect in the human thought process made while trying to understand given information, solve problems, or to use methods and tools. In the context of software requirements specifications, an error is a basic misconception of the actual needs of a user or customer.*
> *Fault – concrete manifestation of an error within the software. One error may cause several faults, and various errors may cause identical faults.*
> *Failure – departure of the operational software system behavior from user expected requirements. A particular failure may be caused by several faults and some faults may never cause a failure.*

We realize that the term *error* has multiple definitions. In fact, IEEE Standard 610 provides four definitions ranging from an incorrect program condition (sometimes referred to as a *program error*) to a mistake in the human thought process (sometimes referred to as a *human error*) [1]. The definition used in this paper more closely correlates to a *human error* rather than a *program error*.

Most previous quality research has focused on the detection and removal of faults (both early and late in the software lifecycle). This research has examined the cause–effect relationships among faults to develop fault classification taxonomies, which are used in many quality improvement approaches (more details in Section 2.1). Despite these research advancements, empirical evidence suggests that quality is still a problem because developers lack an understanding of the source of problems, have an inability to learn from mistakes, lack effective tools, and do not have a complete verification process [19,28,86]. While fault classification taxonomies have proven beneficial, faults still occur. Therefore, to provide more insight into the faults, research needs to focus on understanding the sources of the faults rather than just the faults themselves. In other words, focus on the errors that caused the faults.

As a first step, we performed a systematic literature review to identify and classify the errors identified by other quality researchers. A *systematic literature review* is a formalized, repeatable process in which researchers systematically search a body of literature to document the state of knowledge on a particular subject. The benefits of performing a systematic review, as opposed to using the more common *ad hoc* approach, is that it provides the researchers with more confidence that they have located as much relevant information as possible. This approach is more commonly used in other fields such as medicine to document high-level conclusions that can be drawn from a series of detailed studies [48,79,92]. To be effective, a systematic review must be driven by an overall goal. In this review, the high level goal is to:

> *Identify and classify types of requirement errors into a taxonomy to support the prevention and detection of errors.*

The remainder of this paper is organized as follows. Section 2 describes existing quality improvement approaches and previous uses of errors. Section 3 gives details about the systematic review process and its application. Sections 4 and 5 report the results of the review. Finally, the conclusions and future work are presented in Section 6.

2. Background

To provide context for the review, Section 2.1 first describes successful quality improvement approaches that focus on faults, along with their limitations. These limitations motivate the need to focus on the source of faults (rather than faults alone) as an input to the software quality process. Then, Section 2.2 introduces the concept of *error abstraction* and describes the literature from other fields that is relevant to this review.

2.1. Existing quality improvement approaches

The NASA Software Engineering Laboratory's (SEL) process improvement mechanisms focuses on packaging software process, product and measurement experience to facilitate faster learning [2,9]. This approach classifies faults from different phases into a taxonomy to support risk, cost and cycle time reduction. Similarly, the Software Engineering Institute (SEI) uses a measurement framework to improve quality by understanding process and product quality [36]. This approach also uses fault taxonomies as a basis for building a checklist that facilitates the collection and analysis of faults and improves quality and learning. The SEL and SEI approaches are two successful examples that represent many fault-based approaches. Even with such approaches, faults still exist. Therefore, a singular focus on faults does not ultimately lead to the elimination of all faults. Faults are only a concrete manifestation, or symptom, of the real problem; and without identifying the source, some faults will be overlooked.

One important quality improvement approach that does focus on errors is root cause analysis. However, due to its complexity and expense, it has not found widespread success [55]. Building on root cause analysis, the orthogonal defect classification (ODC) was developed to provide in-process feedback to developers and

Table 1
Limitations of existing quality improvement approaches.

Focus on faults rather than errors
Inability to counteract the errors at their origin and uncover all faults
Lack of methods to help developers learn from mistakes and gain insights into major problem areas
Inability of defect taxonomies to satisfy certain attributes, e.g., simplicity, comprehensiveness, exclusiveness, and intuitiveness
Lack of a process to help developers identify and classify errors
Lack of a complete verification process

help them learn from their mistakes. The ODC also uses an understanding of the faults to identify cause–effect relationships [16,26]. In addition, fault classification taxonomies have been developed and evaluated to aid in quality improvement throughout the development process [37,43,59].

Empirical evidence suggests that "quantifying, classifying and locating individual faults is a subjective and intricate notion, especially during the requirement phase" [38,53]. In addition, Lawrence and Kosuke have criticized various fault classification taxonomies because of their inability to satisfy certain attributes (e.g., simplicity, comprehensiveness, exclusiveness, and intuitiveness) [54]. These fault classification taxonomies have been the basis for most of the software inspection techniques (e.g., checklist-based, fault-based, and perspective-based) used by developers [10,22,25,38,58,75]. However, when using these fault-based techniques inspectors do not identify all of the faults present. To compensate for undetected faults, various supporting mechanisms that have been added to the quality improvement process (e.g., re-inspections and defect estimation techniques [5,11,34,84,85]) have made significant contributions to fault reduction. However, requirements faults still slip to later development phases. This fault slippage motivates a need for additional improvements to the fault detection process. The use of error information is a promising approach for significantly increasing the effectiveness of inspectors.

Table 1 summarizes the limitations of the quality improvement approaches described in this section. To overcome these limitations, the objective of this review is to identify as many errors as possible, as described by researchers in the literature, and classify those errors so that they are useful to developers. A discussion of how the errors can be used by developers is provided in Section 6 of this paper.

2.2. Background on error abstraction

Lanubile et al. first investigated the feasibility of using error information to support the analysis of a requirements document. They defined *error abstraction* as the analysis of a group of related faults to identify the error that led to their occurrence. After an error was identified, it was then used to locate other related faults. One drawback to this process is that the effectiveness depends heavily on the error abstraction ability of the inspectors [53]. To address this problem, our work augments Lanubile et al.'s work by systematically identifying and classifying errors to provide support to the error abstraction process.

In addition to research reported in the software engineering literature, it is likely that error can be identified by reviewing research from other fields. Given that software development is a human-based activity, phenomena associated with the human mental process and its fallibilities can also cause requirements errors. For example, two case studies report that *human reasons*, i.e., reasons not directly related to software engineering, can contribute to fault and error injection [55,86]. Studies of these human reasons can be found in research from the fields of human cognition, and psychology. Therefore, literature from software engineering and psychology are needed to provide a more comprehensive list of errors that may occur.

3. Research method

Following published guidelines [18,30,48], this review included the following steps:

(1) Formulate a review protocol.
(2) Conduct the review (identify primary studies, evaluate those studies, extract data, and synthesize data to produce a concrete result).
(3) Analyze the results.
(4) Report the results.
(5) Discuss the findings.

The review protocol specified the questions to be addressed, the databases to be searched and the methods to be used to identify, assemble, and assess the evidence. To reduce researcher bias, the protocol, described in the remainder of this section, was developed by one author, reviewed by the other author and then finalized through discussion, review, and iteration among the authors and their research group. An overview of the review protocol is provided here, with a complete, detailed description appearing in a technical report [91].

3.1. Research questions

The main goal of this systematic review was to identify and classify different types of requirement errors. To properly focus the review, a set of research questions were needed. With the underlying goal of providing support to the software quality improvement process, the high-level question addressed by this review was:

What types of requirements errors can be identified from the literature and how can they be classified?

Motivated by the background research and to complement the types of errors identified in the software engineering literature, this review makes use of cognitive psychology research into understanding human errors. Fig. 1 shows that the results of a literature search in software engineering about faults and their causes are

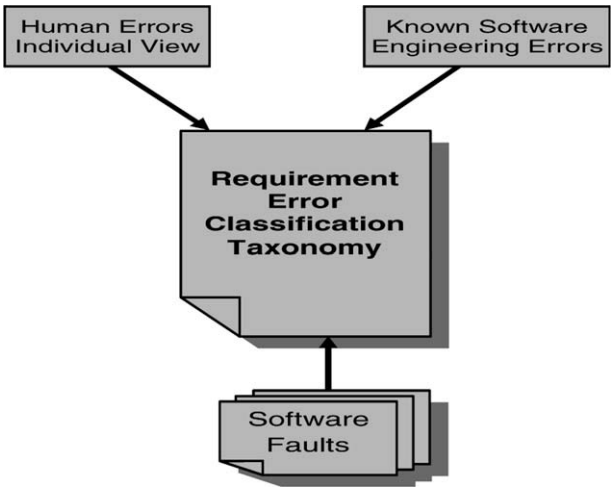

Fig. 1. Types of literature searched to identify the requirement errors.

Table 2
Research questions and motivations.

Research question	Motivation
1. Is there any evidence that using error information can improve software quality? 1.1. Are there any processes or methods reported in literature that use error information to improve software quality? 1.2. Do any of these processes address the limitations and gaps identified in Section 2 (Table 1) of this paper?	Assess the usefulness of errors in existing approaches; identify shortcomings the current approaches and avenues for improvement
2. What types of requirement errors have been identified in the software engineering literature? 2.1. What types of errors can occur during the requirement stage? 2.2. What errors can occur in other phases of the software lifecycle that are related to requirement errors? 2.3. What requirement errors can be identified by analyzing the source of actual faults?	Identify types of errors in the software engineering literature as an input to an error classification taxonomy
3. Is there any research from human cognition or psychology that can propose requirement errors? 3.1. What information can be found about human errors and their classification? 3.2. Which of the human errors identified in question 3.1 can have corresponding errors in software requirements?	Investigate the contribution of human errors from the fields of human cognition and psychology
4. How can the information gathered in response to questions 1–3 be organized into an error taxonomy?	Organize the error information into a taxonomy

combined with the results of a literature search of the more general psychological accounts of human errors to develop the requirement error taxonomy.

The high-level research question was decomposed into three specific research questions and sub-questions shown in Table 2, which guided the literature review. The first question required searching the software engineering literature to identify any quality improvement approaches that focus on errors. These approaches were analyzed to identify any shortcomings and record information about errors. The second question also required searching the software engineering literature with the purpose of explicitly identifying requirement errors and errors from other lifecycle phases (i.e., the design or coding phase) that are related to requirement errors. In addition, this question also required analysis of faults to identify the underlying errors. As a result, this question identified a list of requirement errors from software engineering literature. To answer the third question, we searched the human error literature from the human cognition and psychology fields to identify other types of errors that may occur while developing a requirement artifact. This question involved analyzing the models of human reasoning, planning, and problem solving and their fallibilities to identify errors that can occur during the requirements development stage. Finally, the fourth research question, which is really a meta-question using the information from questions 1 to 3, combines the errors identified from the software engineering literature and the human cognition and psychology literature into a requirement error taxonomy.

3.2. Source selection and search

The initial list of source databases was developed using a detailed source selection criteria as described below:

- The databases were chosen such that they included journals and conferences focusing on: software quality, software engineering, empirical studies, human cognition, and psychology.
- Multiple databases were selected for each research area: software engineering, human cognition, and psychology.
- The databases had to have a search engine with an advanced search mechanism that allowed keyword searches.
- Full text documents must be accessible through the database or through other means.
- Any other source known to be relevant but not included in one of the above databases, were searched separately (e.g., Empirical Software Engineering: An International Journal, text books, and SEI technical reports).
- The list of databases was reduced where possible to minimize the redundancy of journals and proceedings across databases.

This list was then reviewed by a software engineering expert and a cognitive psychology expert who added two databases, a text book and one journal. The final source list appears in Table 3.

To search these databases, a set of search strings was created for each of the three research questions based on keywords extracted from the research questions and augmented with synonyms. Appendix A provides a detailed discussion of the search strings used for each research question.

The database searches resulted in an extensive list of potential papers. To ensure that all papers included in the review were clearly related to the research questions, detailed inclusion and exclusion criteria were defined. Table 4 shows the inclusion and exclusion criteria for this review. The inclusion criteria is specific to each research question, while the exclusion criteria is common for all questions. The process followed for paring down the search results was:

(1) Use the title to eliminate any papers clearly not related to the research focus;
(2) Use the abstract and keywords to exclude papers not related to the research focus; and
(3) Read the remaining papers and eliminate any that do not fulfill the criterion described in Table 4.

After selecting the relevant papers, a quality assessment was performed in accordance with the guidelines published by the Center for Reviews and Discrimination and the Cochrane Reviewers' Handbook as cited by Kitchenham et al. (i.e., each study was assessed for bias, internal validity, and external validity) [48]. First, the study described in each paper was classified as either an experiment or an observational study. Then, a set of quality criteria, focused on study design, bias, validity, and generalizability of results, was used to evaluate quality of the study. The study quality assessment checklist is shown in Table 5.

Using this process, the initial search returned over 25,000 papers, which were narrowed down to 7838 papers based on their titles, then to 482 papers based on their abstracts and keywords.

Table 3
Source list.

Databases	IEEExplore, INSPEC, ACM Digital Library, SCIRUS (Elsevier), Google Scholar, PsychINFO (EBSCO), Science Citation Index
Other journals	Empirical Software Engineering – An International Journal, Requirements Engineering Journal
Additional Sources	Reference lists from primary studies, Books, Software Engineering Institute technical reports

Table 4
Inclusion and exclusion criteria.

RQ	Inclusion criteria	Exclusion criteria
1	• Papers that focus on analyzing/using the errors (source of faults) for improving software quality • Empirical studies (qualitative or quantitative) of using the error information in software lifecycle	• Papers that are based only on expert opinion • Short-papers, introductions to special issues, tutorials, and mini-tracks • Studies not related to any of the research questions • Preliminary conference versions of included journal papers • Studies not in English • Studies whose findings are unclear and ambiguous (i.e., results are not supported by any evidence)
2	• Papers that talk about errors, mistakes or problems in the software development process and requirements in particular • Papers about error, fault, or defect classifications • Empirical studies (qualitative or quantitative) on the cause of software development defects	
3	• Papers from human cognition and psychology about human thought process, planning, or problem solving • Empirical studies on human errors • Papers that survey or describe the human error classifications	

Table 5
Quality assessment.

	Experimental studies	Observational studies
1	Does the evidence support the findings?	Do the observations support the conclusions or arguments?
2	Was the analysis appropriate?	Are comparisons clear and valid?
3	Does study identify or try to minimize biases and other threats?	Does study uses methods to minimize biases and other threats?
4	Can this study be replicated?	Can this study be replicated?

Then, these 482 papers were read to select the final list of 149 papers based on the inclusion and exclusion criteria.

Of these 149 papers, 108 were published in 13 leading journals and six conferences in software engineering, and 41 were published in nine psychology journals. The distribution of the selected papers is shown in Table 6, including the number of papers from each source along with the percentage of the overall total. In addition, three textbooks [69,62,73], provided additional information about human errors that were also found in the journal and conference proceedings included in Table 6. The list of all included literature (journals, conferences proceedings, and textbooks) is provided in Appendix C.

Based on the results from the quality assessment criteria described in Table 5 (and discussed in more detail in Section 6), all of the identified papers identified were of high-quality, providing additional confidence in the overall quality of the set of selected papers. In addition, all of the relevant papers that the authors were aware of prior to the search [16,24,26,28,53,55,69] (i.e., papers that were identified during the background investigation and before commencing the systematic review) were located by the search process during the systematic review, an indication of the completeness of the search. Also, all of the relevant papers from the reference lists of papers identified during the search were found independently by the search process. Contrary to our expectations, Table 6 shows that only a small number of papers appeared in the Journal of Software Testing, Verification, and Reliability, in the International Requirements Engineering Conference, and in the Software Quality Journal. Because we expected these journals to have a larger number of relevant papers, we wanted to ensure there was not a problem with the database being used. Therefore, we specifically searched these two journals again, but no additional relevant material was found.

Furthermore, in some cases a preliminary version of a paper was published in a leading conference with a more complete journal paper following. In this case, only the journal paper was included in the review thereby reducing the number of papers from some conferences (e.g., preliminary work by Sutcliffe et. al., on the impact of human error on system requirements was published in International Conference on Requirements Engineering, but is not included in the review because of a later journal version published in the International Journal of Human–Computer Interaction [80]).

Table 6
Paper distribution.

Source	Count	%
IEEE Computer	19	12.8
Journal of System and Software	13	8.7
Journal of Accident Analysis and Prevention	11	7.4
ACM Transactions on Software Engineering	11	7.4
Communications of the ACM	8	5.4
IBM Systems Journal	8	5.4
IEEE Transaction in Software Engineering	8	5.6
ACM Transactions on Computer–Human Interaction	6	4
Applied Psychology: An International Review	6	4
SEI Technical Report Website	5	3.4
Journal of Information and Software Technology	5	3.4
IEEE Int'l Symposium on Software Reliability Engineering	4	2.7
Journal of Ergonomics	4	2.7
IEEE Trans. on Systems, Man, and Cybernetics (A): Systems and Humans	4	2.7
IEEE International Symposium on Empirical Software Engineering	4	2.7
Requirements Engineering Journal	4	2.7
International Conference on Software Engineering	3	2
Journal of Computers in Human Behavior	3	2
Empirical Software Engineering: An International Journal	3	2
The International Journal on Aviation Psychology	2	1.3
IEEE Annual Human Factors Meeting	2	1.3
International Journal of Human–Computer Interaction	2	1.3
Software Process: Improvement and Practice	2	1.3
Journal of Software Testing, Verification and Reliability	1	0.6
Journal of Reliability Engineering and System Safety	1	0.6
IEEE International Software Metrics Symposium	1	0.6
Journal of Information and Management	1	0.6
Software Quality Journal	1	0.6
High Consequence System Surety Conference	1	0.6
Crosstalk: The Journal of Defense Software Engineering	1	0.6
Journal of IEEE Computer and Control Engineering	1	0.6
Total	149	100

3.3. Data extraction and synthesis

We used data extraction forms to ensure consistent and accurate extraction of the important information from each paper related to the research questions. In developing the data extraction forms, we determined that some of the information was needed regardless of the research question, while other information was specific to each research focus. Table 7 shows the data that was extracted from all papers. Table 8 shows the data that was extracted for each specific research focus.

Table 7
Data items extracted from all the papers.

Data items	Description
Identifier	Unique identifier for the paper (same as the reference number)
Bibliographic	Author, year, title, source
Type of article	Journal/conference/technical report
Study aims	The aims or goals of the primary study
Context	Context relates to one/more search focus, i.e., research area(s) the paper focus on
Study design	Type of study – industrial experiment, controlled experiment, survey, lessons learned, etc.
Level of analysis	Single/more researchers, project team, organization, department
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, discussion, and documents
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 studies

Table 8
Data items extracted for each research 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", how does it relate to our research
Requirement errors	Problems	Problems found in requirement stage
	Errors	Whether the problems constitute requirement stage errors
	Faults	Faults at requirement stage and their causes
	Mechanism	Process used to analyze or abstract requirement errors
Error–fault–defect taxonomies	Focus	The focus of the taxonomy (i.e., error, fault, or failure)
	Error focus	Yes or No; if "Yes", how does it relate to our research questions
	Requirement phase	Yes or No (whether it was applied in requirement phase)
	Benefits	Benefits of the taxonomy
	Limitation	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 does it relate to our research questions
	Requirement phase	Yes or No (Did it inspect requirement documents?)
	Benefits	Benefits of the inspection method
	Limitation	Limitations of the 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
	Errors attributable	Software development errors that can be attributable to human errors
	Related faults	Software faults that can be caused by human errors
	Evidence	The empirical evidence regarding errors made by humans in different situations (e.g., aircraft control) that are related to requirement errors

Consistent with the process followed in previous systematic reviews (e.g., [49]), the first author reviewed all papers and extracted data. Then, the second author independently reviewed and extracted data from a sample of the papers. We then compared the data extracted by each reviewer for consistency. We found that we had consistently extracted information from the sample of papers, suggesting that the second author did not need to review the remainder of papers in detail and that the information extracted by the first author was sufficient. The data extracted from all papers was synthesized to answer each question as described in Section 4.

4. Reporting the review

4.1. Question 1: Is there any evidence that using error information can improve software quality?

The idea of using the source of faults to improve software quality is not novel. Other researchers have used this information in different ways with varying levels of success. A review of the different methods indicates that knowledge of the source of faults is useful for process improvement, defect prevention by helping developers learn from their mistakes and detection of defects during inspections. A major drawback of these approaches is that they typically do not provide a formal process to assist developers in finding and fixing errors. In fact, only the approach by Lanubile et al. provided any systematic way to use error information to improve the quality of a requirements document [53]. Another drawback to these methods is that they rely only on a sample of faults to identify errors, therefore potentially overlooking some errors.

While these approaches have positive aspects and describe software errors (as discussed in the answers to 1.1), they also have limitations (as discussed in the answers to 1.2). A total of 13 papers addressed this question and its related sub-questions.

4.2. Question 1.1: Are there any processes or methods reported in literature that use error information to improve software quality?

The review identified nine methods that stress the use of error information during software development. These methods are de-

scribed in this section along with their use in software organizations, and the types of errors they helped identify.

- The defect causal analysis approach is a team-based quality improvement technique used to analyze samples of previous faults to determine their cause (the error) in order to suggest software process changes and prevent future faults. Empirical findings show that the use of the defect causal analysis method at IBM and at Computer Sciences Corporation (CSC) resulted in a 50% decrease in defects over each two year period studied. Based on the results from these studies, Card suggests that all causes of software faults fall into one of four categories: (a) methods (incomplete, ambiguous, wrong, or enforced), (b) tools and environment (clumsy, unreliable, or defective), (c) people (who lack adequate training or understanding), and (d) input and requirements (incomplete, ambiguous, or defective) [24].
- The software failure analysis approach is a team-based quality approach. The goal of this process is to improve the software development and maintenance process. Using this approach, developers analyze a representative sample of defect data to understand the causes of particular classes of defects (i.e., specification, user-interface, etc.). This process was used by engineers at Hewlett–Packard to understand the cause of user-interface defects and develop better user interface design guidelines. During their next year-long project, using these guidelines, the user-interface defects found during test decreased 50%. All the causes of user-interface defects were classified into four different categories: guidelines not followed, lack of feedback or guidelines, different perspectives, or oops! (forgotten) [41].
- The defect causal analysis (using experts) approach accumulates expert knowledge to substitute for an in-depth, per-object defect causal analysis. The goal is to identify the causes of faults found during normal development, late in the development cycle and after deployment. Based on the results of a study, Jacobs et al. describe the causes of software defects, e.g., inadequate unjustified trust, inadequate communication, unclear responsibilities, no change control authority, usage of different implementations, etc. The study also showed that the largest numbers of defects were the result of communication problems [46].
- The defect prevention process uses causal analysis to determine the source of a fault and to suggest preventive actions. Study results show that it helps in preventing commonly occurring errors, provides significant reductions in defect rates, leads to less test effort, and results in higher customer satisfaction. Based on the analysis of different software products over six years, Mays et al. classified defect causes as oversight causes (e.g., developer overlooked something, or something was not considered thoroughly), education causes (developer did not understand some aspect), communication causes (e.g., something was not communicated), or transcription causes (developer knew what to do but simply made a mistake) [60].
- The software bug analysis process identifies the source of bugs. It also contains countermeasures (with implementation guidance) for preventing bugs. To assess the usefulness of this approach, a sample of 28 bugs found during the debug and test phases of authorization terminal software development were analyzed by group leaders, designers, and third party designers to determine their cause. A total of 23 different causes were identified, and more than half of these causes were related to designers carelessness [61].
- The root cause analysis method introduced the concept of a multi-dimensional defect trigger to help developers determine the root cause of a fault (the error) in order to identify areas for process improvement. Leszak et al. conducted a case-study of the defect modification requests during the development of a transmission network element product. They found the root causes of defects to include: communication problems or lack of awareness of the need for communication, lack of domain/system/tools/process knowledge, lack of change coordination, and individual mistake [55].
- The error abstraction process analyzes groups of related faults to determine their cause (i.e., the error). This error information is then used to find other related faults in a software artifact [53].
- The defect based software process improvement analyzes faults through attribute focusing to provide insight into their potential causes and makes suggestions that can help a team adjust their process in real time [24,59].
- The goal-oriented process improvement methodology also uses defect causal analysis for tailoring the software process to address specific project goals in a specific environment. Basili and Rombach describe an error scheme that classifies the cause of software faults into different problem domains. These problem domains include: application area (i.e., misunderstanding of application or problem domain), methodology to be used (not knowing, misunderstanding, or misuse of problem solution processes), and the environment of the software to be developed (misunderstanding or misuse of the hardware or software of a given project) [8].
- Total quality management is a philosophy for achieving long-term success by linking quality with customer satisfaction. Key elements of this philosophy include total customer satisfaction, continuous process improvement, a focus on the human side of quality, and continuous improvement for all quality parameters. It involves identifying and evaluating various probable causes (errors) and testing the effects of change before making it [47]. This approach is based on the defect prevention process described earlier.

Each of the above methods uses error information to improve software quality. Each has shown some benefits and identified some important types of errors. This information served as an input to the requirement error taxonomy.

4.3. Question 1.2: Do any of these processes address the limitations and gaps identified in Section 2 of this paper?

Each of the methods described in the answer to *question 1.1* does address many of the limitations described in Table 1, but, they still have some additional limitations. The limitations common to the different methods are summarized in Table 9.

While these methods do analyze the source of faults, they only use a small sample of faults, potentially overlooking some errors. As a result, the error categories are generic and appear to be incomplete. Moreover, some methods only describe the cause categories (and not actual errors), while others provide only a few examples for each category. None of the methods provides a list of all the errors that may occur during the requirements phase. Another common problem is that while many methods do emphasize the importance of *human errors* and provide a few examples (e.g., errors due to designers carelessness and transcription errors), they do not go far enough. These approaches lack a strong cognitive theory to describe a more comprehensive list of the errors. Finally, no formal process is available that can guide developers in identifying the errors and resulting faults when they occur. The inability of the

Table 9
Limitations of methods in question 1.1.

Limitations	Methods
Requires extensive documentation of problem reports and inspection results; and over-reliance on historical data	[24,50,47,55,59,60]
Cost of performing the causal analysis through implementing actions ranges from 0.5% to 1.5% of the software budget; and requires a startup investment	[24,41,59,60]
It is cost-intensive, people-intensive and useful for analyzing only a small sample of faults, or group of related faults	[8,24,41,46,59–61]
Requires experienced developers, extensive meetings among experts, and interviewing the actual developers to analyze the probable causes of faults	[46]
Results in a large number of actions, and has to applied over a period of time to test the suggested actions/improvements to software process	[55,59–61]
It can only detect the process inadequacy/process improvement actions and not reveal the actual error	[47,59–61]
It analyzes the faults found late in the software lifecycle	[46,59–61]
Does not guide the inspectors, and relies heavily on the creativity of inspectors in abstracting errors from faults during software inspections	[53]

existing methods to overcome these limitations motivates the need for development of more complete requirement error taxonomy.

4.4. Question 2: What types of requirement errors have been identified in the software engineering literature?

The identification of requirement errors will support future research into software quality. To obtain an initial list of errors, the software engineering literature was reviewed to extract any errors that had been described by other researchers. This research question is addressed in detail by sub-questions 2.1 and 2.2. A total of 55 papers were analyzed to identify the types of requirement errors in software engineering literature, 30 to address question 2.1 and 24 to address question 2.2. The complete list of all the requirement errors that were identified in the software engineering literature is provided in Appendix B.

4.5. Question 2.1: What types of errors can occur during the requirement stage?

The review uncovered a number of candidates for requirement errors. Table lists potential requirement errors along with the relevant references.

The first row of Table 10 contains six different studies that analyzed faults found during software development using different causal analysis methods (as described in question 1.1) to identify requirement errors [16,24,41,46,60,61]. Examples of the errors identified by these studies include: lapses in communications among developers, inadequate training, not considering all the ways a system can be used, and misunderstanding certain aspect of the system functionality.

The second source of errors includes empirical studies that classify requirement errors problems experienced by developers, studies that classify difficulties in determining requirements, and other case studies that trace requirement errors to the problems faced during the requirements engineering process [13,21,42,77,81,82]. Examples of the errors from these studies include: lack of user participation, inadequate skills and resources, complexity of application, undefined requirement process, and cognitive biases.

The third source of errors include studies that describe the root causes (some of whom were those belonging to requirement stage) of troubled IT projects [7,39,76]. Examples of the errors from these studies include: unrealistic business plan, use of immature technology, not involving user at all stages, lack of experienced or capable management, and improper work environment.

Table 10 also includes the findings from an empirical study of 13 software companies that identified 32 potential problem factors, and other similar empirical studies that identified requirement problem factors affecting software reliability [12,27,79,92]. Examples of the errors from these studies include: hostile team relationships, schedule pressure, human nature (mistakes and omissions), and flawed analysis method.

Table 10 also includes studies that determined the causes of requirement traceability and requirement inconsistencies, errors due to domain knowledge, requirement management errors, and errors committed among team members during software development [29,33,64,71]. Examples of the errors from these studies include: poor planning, insufficiently formalized change requests, impact analysis not systematically achieved, misunderstandings due to working with different systems, and team errors.

Finally, Table 10 contains "*other errors*" that includes studies describing the root causes of safety-related software errors and description of requirement errors causing safety-related errors as well as studies that describe requirement errors based on the contribution of human error from social and organizational literature [56,57,87]. Examples of the errors from these studies include misunderstanding of assumptions and interface requirements, misunderstanding of dependencies among requirements, mistakes during application of well-defined process, and attention or memory slips. The complete list of requirement errors derived from the sources listed in Table 10 is shown in Appendix B.

Table 10
Sources used to identify requirement errors in the literature.

Sources of errors	References
Root causes, cause categories, bug causes, defect–fault causes	[16,24,41,46,60,61]
Requirement engineering problem classification	[13,21,42,77,81,82]
Empirical studies on root causes of troubled projects or errors	[7,39,76]
Influencing factors in reference stage and software development	[12,27,79,92]
Causes of requirement traceability and requirement inconsistency	[29,33,65,71]
Domain knowledge problems	[65]
Management problems	[29]
Situation awareness/decision making errors	[33]
Team errors	[71]
Other	[40,56,57,87]

4.6. Question 2.2: What errors can occur in other phases of the software lifecycle that are related to errors that can occur during the requirements phase?

In addition to the errors found specifically in the requirements phase, the review also uncovered errors that occur during the design and coding phases. These errors can also occur during the requirements phase, including:

- *Missing information:* Miscommunication between designers in different teams; lack of domain, system, or environmental knowledge; or misunderstandings caused by working simultaneously with several different software systems and domains [17,41].
- *Slips in system design:* Misunderstanding of the current situation while forming a goal (resulting in an inappropriate choice of actions), insufficient specification of actions to follow for achieving goal (resulting in failure to complete the chosen actions),

using an analogy to derive a sequence of actions from another similar situation (resulting in the choice of a sequence of actions that is different from what was intended), or the sequence of actions is forgotten because of an interruption [63,64].
- *System programs:* Technological, organizational, historical, individual, or other causes [31].
- *Cognitive breakdown:* Inattention, over attention, choosing the wrong plan due to information overload, wrong action, incorrect model of problem space, task complexity, or inappropriate level of detail in the task specification [50,51].

4.7. Question 2.3: What requirement errors can be identified by analyzing the source of actual faults?

In addition to identifying requirement errors directly reported in the literature, this review also identified a list of faults that could be traced back to their underlying error [3,4,6,14,15,26,37,44,68,70,72]. For example, in the case where a requirements document is created by multiple sub-teams, a particular functionality is missing because each sub-team thought the other one would include it. The error that caused this problem is the lack of communication among the sub-teams. Another example is the case where two requirements are incomplete because they are both missing important information about the same aspect of system functionality. The error that caused this problem is the fact that developers did not completely understand that aspect of the system and it was manifested in multiple places. These errors were added to the list of errors that served as input to *question 4*. These errors include:

- Misunderstanding or mistakes in resolving conflicts (e.g., there are unresolved requirements or incorrect requirements that were agreed on by all parties).
- Mistakes or misunderstandings in mapping inputs to outputs, input space to processes, or processes to output.
- Misunderstanding of some aspect of the overall functionality of the system.
- Mistakes while analyzing requirement use cases or different scenarios in which the system can be used.
- Unresolved issues about complex system interfaces or unanticipated dependencies.

4.8. Question 3: Is there any research from human cognition or psychology that can propose requirement errors?

To address the fact that requirements engineering is a human-based activity and prone to errors, the review also examined human cognition and psychology literature. The contributions of this literature to requirements errors are addressed by sub-questions 3.1 and 3.2. A total of 32 papers that analyzed the human errors and their fallibilities were used to identify errors that may occur during software requirements phase.

4.9. Question 3.1: What information can be found about human errors and their classification?

The major types of human errors and classifications identified in human cognition and psychology include:

- *Reason's classification of mistakes, lapses, and slips:* Errors are classified as either mistakes (i.e., the wrong plan is chosen to accomplish a particular task), lapses (i.e., the correct plan is chosen, but a portion is forgotten during execution), or slips (i.e., the plan is correct and fully remembered, but during its execution something is done incorrectly) [23,31,64].
- *Rasmussen's skill, rule and knowledge based human error taxonomy:* Skill based slips and lapses are cause by mistakes while executing a task even though the correct task was chosen. Rule and knowledge based mistakes occur due to errors in intentions, including choosing the wrong plan, violating a rule, or making a mistake in an unfamiliar situation [23,63,40].
- *Reason's general error modeling system (GEMS): A* model of human error in terms of unsafe acts that can be intentional or unintentional. Unintentional acts include slips and lapses while intentional acts include mistakes and violations [23,35,63,78].
- *Senders and Moray's classification of phenomenological taxonomies, cognitive taxonomies, and deep rooted tendency taxonomies: Description* of the how, what and why concerns of an error, including omissions, substitutions, unnecessary repetitions, errors based on the stages of human information processing (e.g., perception, memory, and attention), and errors based on biases [23,78].
- *Swain and Guttman's classification of individual discrete actions:* Omission errors (something is left out), commission errors (something is done incorrectly), sequence errors (something is done out of order), and timing errors (something is done too early or too late) [83].
- *Fitts and Jones control error taxonomy:* Based on a study of "pilot error" that occur while operating aircraft controls. The errors include: substitution (choosing the wrong control), adjustment (moving the control to the wrong position), forgetting the control position, unintentional activation of the control, and inability to reach the control in time [35].
- *Cacciabue's taxonomy of erroneous behavior:* Includes system and personnel related causes. Errors are described in relation to execution, planning, interpretation, and observation along with the correlation between the cause and effects of erroneous behavior [23].
- *Galliers, Minocha, and Sutcliffe's taxonomy of influencing factors for occurrence of errors: Environmental* conditions, management and organizational factors, task/domain factors, and user/personnel qualities including the slip and mistake types of errors described earlier [40].
- *Sutcliffe and Rugg's error categories: Operational* description, cognitive causal categories, social and organizational causes, and design errors [81].
- *Norman's classification of human errors: Formation* of intention, activation, and triggering. Important errors include errors in classifying a situation, errors that result from ambiguous or incompletely specified intentions, slips from faulty activation of schemas, and errors due to fault triggering [62–64,66].
- *Human error identification (HEI) tool:* Describes the SHERPA tool that classifies errors as action, checking, retrieval, communication, or selection [74,78].
- *Human error reduction (HERA):* Technique that analyzes and describes human errors in air the traffic control domain. HERA contains error taxonomies for five cognitive domains: perception and vigilance, working memory, long-term memory, judgment, planning and decision-making, and response execution [20,45].

4.10. Question 3.2: Which of the human errors identified in question 3.1 can have corresponding errors in software requirements?

Those errors that were relevant to requirements were included in the initial list of errors that served as input to *question 4* to make it more comprehensive. Examples of the translation of these errors into requirements errors are found in Table 11.

Table 11
Requirement errors drawn from human errors.

Error	Description
Not understanding the domain	Misunderstandings due to the complex nature of the task; some properties of the problem space are not fully investigated; and, mistaken assumptions are made
Not understanding the specific application	Misunderstanding the order of events, the functional properties, the expression of end states, or goals
Poor execution of processes	Mistakes in applying the requirements engineering process, regardless of its adequacy; out of order steps; and lapses on the part of the people executing the process
Inadequate methods of achieving goals and objectives	System-specific information was omitted leading to the selection of the wrong technique or process; selection of a technique or process that, while successful on other projects, has not been fully investigated or understood in the current situation
Incorrectly translating requirements to written natural language	Lapses in organizing requirements; omission of necessary verification at critical points during the execution of an action; repetition of verification leading to the repetition or omission of steps
Other human cognition errors	Mistakes caused by adverse mental states, loss of situation awareness, lack of motivation, or task saturation; mistakes caused by environmental conditions

4.11. Question 4: How can the information gathered in response to questions 1–3 be organized into an error taxonomy?

Some of the errors were identified in more than one of the bodies of literature surveyed: quality improvement approaches, requirement errors, other software errors, and human errors. The errors described in the answers to questions 1–3 were collected, analyzed and combined into an initial requirement error taxonomy with the objective of making the taxonomy simple and easy to use yet comprehensive enough to be effective.

The development of the requirement error taxonomy consisted of two step. First, the detailed list of errors identified during the literature search, and described in the answers to questions 1–3, were grouped together into a set of detailed requirement error classes. Second, the error classes were grouped into three high-level requirement error types. These high-level error types were also found during the literature search. The next section details the process of developing the requirement error taxonomy and how the taxonomy was constructed.

5. Process of developing the requirement error taxonomy

We organized the errors into a taxonomy that will guide future error detection research with the intent of addressing limitations in the existing quality improvement approaches. The errors identified from the software engineering and psychology fields were collected, analyzed for similarities, and grouped into the taxonomy. Errors that had similar characteristics (symptoms), whether from software engineering or from psychology, were grouped into an error class to support the identification of related errors when an error is found. An important constraint while grouping the requirement errors was to keep the error classes as orthogonal as possible [87,88].

In Section 5.1, for each error class, we first describe the error class. Then we provide a table that lists the specific errors from the literature search that were grouped into that error class. Finally, we give an example of an error from that class, along with a fault likely to be caused by that error. The examples are drawn from two sample systems. Due to space limitations, only a brief overview of each example system is provided here.

- *Loan arranger system (LA):* The LA application supports the business of a loan consolidation organization. This type of organization makes money by purchasing loans from banks and then reselling those loans to other investors. The LA allows a loan analyst to select a bundle of loans that have been purchased by the organization that match the criteria provided by an investor. This criterion may include amount of risk, principal involved and expected rate of return. When an investor specifies investment criteria, the system selects the optimal bundle of loans that satisfy the criteria. The LA system automates information management activities, such as updating loan information provided monthly by banks.
- *Automated ambulance dispatch system (AAD):* This system supports the computer-aided dispatch of ambulances to improve the utilization of ambulances and other resources. The system receives emergency calls, evaluates incidents, issues warnings, and recommends ambulance assignments. The system should reduce the response time for emergency incidents by dispatching decisions based on recommendations made by system [4,52].

5.1. Developing the requirement error classes

The requirement error classes were created by grouping errors with similar characteristics. Tables 12–25 show each error class along with the specific errors (from Software Engineering and Human Cognition fields) that make up that class.

Table 12
Communication errors.

Inadequate project communications	[41]
Changes in requirements not communicated	[46]
Communication problems, lack of communication among developers and between developers and users	[60]
Communication problems	[55]
Poor communication between users and developers, and between members of the development team	[47]
Lack of communication between sub-teams	[16]
Communication between development teams	[13]
Lack of user communication	[13]
Unclear lines of communication and authority	[39]
Poor communication among developers involved in the development process	[12,27]
Communication problems, information not passed between individuals	[33]
Communication errors within a team or between teams	[56]
Lack of communication of changes made to the requirements	[56]
Lack of communication among groups of people working together	[71]

Table 13
Participation errors.

No involvement of all the stakeholders	[46]
Lack of involvement of users at all times during requirement development	[47]
Involving only selected users to define requirements due to the internal factors like rivalry among developers or lack of the motivation	[39,40,87]
Lack of mechanism to involve all the users and developers together to resolve the conflicting requirements needs	[24]

Table 14
Domain knowledge errors.

Lack of domain knowledge or lack of system knowledge	[12,17,40,41,55,92]
Complexity of the problem domain	[12,13,21,42]
Lack of appropriate knowledge about the application	[27]
Complexity of the task leading to misunderstandings	[40]
Lack of adequate training or experience of the requirement engineer	[41]
Lack of knowledge, skills, or experience to perform a task	[71]
Some properties of the problem space are not fully investigated	[31]
Mistaken assumptions about the problem space	[56]

Table 15
Specific application errors.

Lack of understanding of the particular aspects of the problem domain	[60,65]
Misunderstandings of hardware and software interface specification	[56]
Misunderstanding of the software interfaces with the rest of the system	[56]
User needs are not well-understood or interpreted while resolving conflicting requirements	[61]
Mistakes in expression of the end state or output expected	[64]
Misunderstandings about the timing constraints, data dependency constraints, and event constraints	[56,57]
Misunderstandings among input, output, and process mappings	[70]

Table 16
Process execution errors.

Mistakes in executing the action sequence or the requirement engineering process, regardless of its adequacy	[23,35,63,78]
Execution or storage errors, out of order sequence of steps and slips/lapses on the part of people executing the process	[35,40,66,85]

Table 17
Other human cognition errors.

Mistakes caused by adverse mental states, loss of situation awareness	[33,71,87]
Mistakes caused by ergonomics or environmental conditions	[12]
Constraints on humans as information processors, e.g., task saturation	[40]

Table 18
Inadequate method of achieving goals and objectives errors.

Incomplete knowledge leading to poor plan on achieving goals	[40]
Mistakes in setting goals	[40]
Error in choosing the wrong method or wrong action to achieve goals	[50,51]
Some system-specific information was misunderstood leading to the selection of wrong method	[33]
Selection of a method that was successful on other projects	[23,69]
Inadequate setting of goals and objectives	[76]
Error in selecting a choice of a solution	[56]
Using an analogy to derive a sequence of actions from other similar situations resulting in the wrong choice of a sequence of actions	[31,40,64,69]
Transcription error, the developer understood everything but simply made a mistake	[60]

Table 19
Management errors.

Poor management of people and resources	[27,71]
Lack of management leadership and necessary motivation	[29]
Problems in assignment of resources to different tasks	[13]

The communication errors class (Table 12) describes errors that result from poor or missing communications among the various stakeholders involved in developing the requirements.

Table 20
Requirement elicitation errors.

Inadequate requirement gathering process	[16]
Only relying on selected users to accurately define all the requirements	[39]
Lack of awareness of all the sources of requirements	[27]
Lack of proper methods for collecting requirements	[13]

Table 21
Requirement analysis errors.

Incorrect model(s) while trying to construct and analyze solution	[50,51]
Mistakes in developing models for analyzing requirements	[8]
Problem while analyzing the individual pieces of the solution space	[13]
Misunderstanding of the feasibility and risks associated with requirements	[87]
Misuse or misunderstanding of problem solution processes	[8]
Unresolved issues and unanticipated dependencies in solution space	[17]
Inability to consider all cases to document exact behavior of the system	[60]
Mistakes while analyzing requirement use cases or scenarios	[8,74]

Table 22
Requirement traceability errors.

Inadequate/poor requirement traceability	[13,42,76]
Inadequate change management, including impact analysis of changing requirements	[29]

Table 23
Requirement organization errors.

Poor organization of requirements	[41]
Lapses in organizing requirements	[23]
Ineffective method for organizing together the requirements documented by different developers	[8]

Table 24
No use of standard for documenting errors.

No use of standard format for documenting requirements	[41]
Different technical standard or notations used by sub teams for documenting requirements	[39]

Table 25
Specification errors.

Missing checks (item exists but forgotten)	[16]
Carelessness while organizing or documenting requirement regardless of the effectiveness of the method used	[7,8]
Human nature (mistakes or omissions) while documenting requirements	[83,92]
Omission of necessary verification checks or repetition of verification checks during the specification	[50,51]

Example:

Error: Customer did not communicate that the LA system should be used by between one and four users (loan analysts) simultaneously.

Fault: Omitted functionality because the requirements specify operations as if they are performed by only one user at a time.

The participation error class (Table 13) describes errors that result from inadequate or missing participation of important stakeholders involved in developing the requirements.

Example:

Error: Bank lender, who was not involved in the requirements process, wanted the LA application system to handle both fixed rate loans and adjustable rate loans.

Fault: Omitted functionality as requirements only consider fixed rate loans.

The domain knowledge error class (Table 14) describes errors that occur when requirement authors lack knowledge or experience about the problem domain.

Example:

Error: Requirement author lacks knowledge about the relative priority of emergency types within the AAD domain.

Fault: The functionality is incorrect because the requirements contain the wrong ambulance dispatch algorithm.

The specific application error class (Table 15) describes errors that occur when the requirement authors lack knowledge about specific aspects of the application being developed (as opposed to the general domain knowledge).

Example:

Error: Requirement author does not understand the order in which status changes should be made to the ambulances in the AAD system.

Fault: The requirements specify an incorrect order of events, the status of the ambulance is updated after it is dispatched rather than before, leaving a small window of time in which the same ambulance could be dispatched two times.

The process execution error class (Table 16) describes errors that occur when requirement authors make mistakes while executing the requirement elicitation and development processes regardless of the adequacy of the chosen process.

Example:

Error: Misunderstanding of the ordering of the transactions involving the creation or deletion of records in the database for the LA system.

Fault: The requirements incorrectly specify how the LA should handle transactions. They state that transactions should be resolved based on the order in which the processing completes rather than in the order in which the requests were received.

The other human cognition error class (Table 17) describes other errors that result from constraints on the cognitive abilities of the requirement authors.

The inadequate method of achieving goals and objective error class (Table 18) describes errors that result from selecting inadequate or incorrect methods for achieving the stated goals and objectives.

Example:

Error: The requirements engineering misunderstood that some crucial functionality had to be delivered before other functionality, so s/he chose the waterfall lifecycle rather than an incremental one.

Fault: Required functionality cannot be delivered on time to the customer.

The management error class (Table 19) describes errors that result from inadequate or poor management processes.

Example:

Error: In the LA system, the same requirement engineer is assigned to document the borrower's risk requirement and the loan's risk requirements. These contrasting tasks result in a mental lapse when understanding the inputs required to successfully produce the risk estimates.

Fault: The functionality for calculating risk is incorrect.

The requirement elicitation error class (Table 20) describes errors that result from the use of an inadequate requirement elicitation process.

Example:

Error: In the AAD system, the requirements engineers are not able to elicit requirements about system response time for emergency incidents or about error handling.

Fault: Performance and other non-functional requirements are omitted.

The requirement analysis error class (Table 21) describes errors committed during the requirement analysis process.

Example:

Error: In the LA system, the analysis process was not able to identify how the system should respond if multiple transactions are requested that include the same loan before the system is updated.

Fault: The requirements omit this situation leaving it undefined and possibly erroneous.

The requirement traceability error class (Table 22) describes that result from an inadequate or incomplete requirement traceability process.

Example:

Error: In the LA system, a requirement describing the ability of loan analyst to change the borrower information can not be traced to any user need.

Fault: An extraneous requirement is included that could result in extra, unnecessary work for the developers.

The requirement organization error class (Table 23) describes errors committed while organizing the requirement during the documentation process.

Example:

Error: When creating the requirements document for the LA system, the requirement engineer does not use any type of logical organization of the requirements.

Fault: Because the requirements are not grouped logically, a requirement about how to display the results of a report was omitted from the requirements document.

The no use of documentation standard error class (Table 24) describes errors committed because the requirement author did not use a standard for documenting the requirements.

Example:

Error: In documenting the LA application system, the IEEE Standard was not used.

Fault: Requirements about system scope and performance were omitted.

The specification error class (Table 25) describes general errors that can occur while specifying the requirements regardless of whether the developers correctly understood the requirements.

Example:

Error: In the LA application system, the requirement author understood the difference between regular loans (i.e., for amount $\leqslant$\$275,000) and jumbo loans (i.e., for amount >\$275,000), but while documenting the requirements, s/he recorded the same information for both type of loans.

Fault: The requirements for the jumbo loans incorrectly specify exactly the same behavior as for regular loans.

5.2. Development of requirement error types

In order to make the taxonomy understandable and usable for developers, it includes two levels. The taxonomy organizes the er-

ror classes, from Section 5.1, into three high-level types: *people errors*, *process errors*, and *documentation errors*. These high-level types can be found in the literature. The remainder of this section describes the three error types and the distribution of the error classes over three error types.

Table 26
Requirement error types.

Reference	Error types
Card [24]	1. Incomplete or wrong *methods* 2. Clumsy or defective *tools and environment* 3. *People* who lack adequate training or understanding 4. Incomplete or defective *input and requirement*
Jacobs et al. [46]	1. Inadequate communication 2. Process risks 3. Organizational risks concerning decision authorities, responsibilities 4. Technology risks concerning methods and toolsDoes not mention people risks as a separate type, but are embedded in other risk types
Mays et al. [60]	1. Oversight of the developer 2. Education of the developer 3. Communication failure 4. Transcription error (developer knew what to do and understood but simply made a mistake)
Basili and Rombach [8]	1. Application errors (misunderstanding of the problem domain) 2. Problem–solution errors (not knowing, misunderstanding, or misuse of problem solution processes) 3. Environment errors (misunderstanding of the hardware or software environment) 4. Information management errors (mishandling of certain procedures) 5. Clerical errors (carelessness)
Beecham et al. [13]	1. Organizational issues (i.e., developer communication, culture, resources, skills, training, user communication) 2. Technical issues (i.e., complexity of the application, poor user understanding, requirements traceability, undefined requirement process)
Sutcliffe et al. [81]	1. Communication problems 2. Social and organization problems (i.e., user participation, lack of understanding, or trained personnel) 3. Politics 4. Technical problems
Coughlan and Macredie [27]	1. Poor communication 2. Lack of appropriate knowledge or shared understanding 3. Inappropriate, incomplete, or inaccurate documentation 4. Lack of systematic process 5. Poor management of people or resources
Lutz [56]	1. Program faults (documented software errors) 2. Human errors (i.e., communication errors, misunderstandings specifications, or problem domain) 3. Process flaws (i.e., flaws in controlling system complexity or development methods)

The different high-level error types found in the literature are shown in Table 26, along with their source. Various researchers have described different classifications of requirement errors. We have based our error types on this information as described below.

A major focus of the research in Table 26 is the contribution of *people* to the errors. Some researchers have explicitly labeled them as people errors or human errors (i.e., Card et al. [24] and Lutz et al. [56]), while others mentioned the error types that are more detailed variants of the people errors (e.g., inadequate communication, errors due to the education of a developer, misunderstandings, etc.). These errors are already described in the requirement error classes shown in Section 5.1.

Another focus in Table 26 concerns the errors related to *process* flaws. Jacobs et al. [40] and Coughlan et al. [27] have mentioned errors due to flawed development process and flawed methods and tools. While other researchers have recognized this source of errors, they often combine process errors with other error types. An important aspect of our taxonomy is that we distinguish between flaws in the process (i.e., process errors) and human contribution to the error (i.e., people errors).

Mays et al. [60] also listed transcription errors, i.e., the developer knew what to do and understood but simply made a mistake. Based on this idea, we developed the *documentation* error type to describe errors caused by mistakes in organizing and specifying the requirements regardless of whether the developer understood the requirement.

Therefore, the error types used in our requirement error taxonomy are: *people errors*, *process errors*, and *documentation errors*. *People errors* include errors that originate with fallibilities of the individuals involved in project development. *Process errors* are caused by mistakes in selecting the means of achieving goals and objectives and focus mostly on the inadequacy of the requirement engineering process. *Documentation errors* are caused by mistakes in organizing and specifying the requirements irrespective of whether the developer understood the requirements. After identifying the 14 error types and their meaning, the next step was to classify them using these three error types. Fig. 2 shows results of that classification as the full requirement error taxonomy.

We include the *communication* and *participation* error classes in the people error type (because they deal with problems among people involved in the development). The *domain knowledge* and *specific application* error classes are included in the people error type because they relate to the lack of adequate training provided to people or a person's misunderstanding of the problem domain. *Process execution* errors are included in the people error type because they relate to errors committed by people during the execution of processes, rather than errors associated with the adequacy

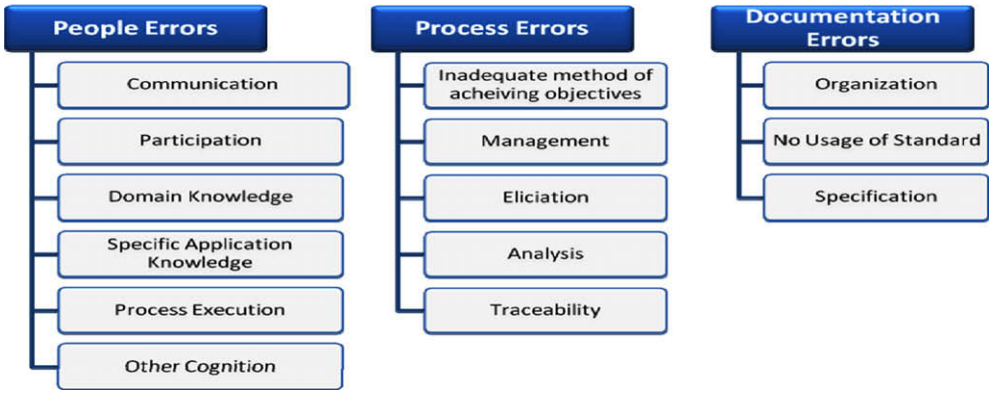

Fig. 2. Requirement error taxonomy.

Table 27
Excluded requirement errors.

Error	Exclusion reason
Culture, staff retention, politics	Based only on expert opinion
Inappropriate, hidden or missing functionality, poor feedback, syntax and semantic errors	Not related to any of the research questions
Work standard, getting slow start, relying only on user interviews, time pressure, situation awareness errors, lack of trust	Short-papers, introductions to special issues, tutorials, and mini-tracks

of the process itself. Finally, *other human cognition* errors are included in the people error type because they describe errors that result from constraints on the cognitive abilities of people.

The process error type includes the error classes that described the errors resulting from flawed methods and tools used for achieving goals and flawed requirement engineering processes. The error classes included in process error were: *inadequate method for achieving goals and objectives*, *requirement management* errors, *requirement elicitation* errors, *requirement analysis* errors, and *requirement traceability* errors.

Finally, the documentation error type includes errors that result from the process of documenting the information about the requirements. They do not concern misunderstandings of the actual requirements, those are people errors. The three error types that are related to this problem are *specification errors*, *no usage of standard errors*, and *organization errors*.

Finally, there were some errors that were found during the systematic review that were excluded from the requirement error taxonomy. Table 27 lists those errors along with the reasons for excluding them.

6. Discussion

This section summarizes the principal findings of the systematic review, highlights the strengths and weaknesses of the evidence gathered and discusses the relevance and contribution of the *requirement error taxonomy* to the software engineering research and practice community.

6.1. Principal findings

The goal of this review was to identify and classify errors that can occur during the requirement phase. Thus, a systematic literature review was conducted covering various research domains. Using this information, a comprehensive initial requirement error taxonomy was developed. The principal findings of the review are:

- A description of software quality improvement methods that use error information, their limitations, drawbacks, and contributions to the requirements error taxonomy.
- A description of the requirements errors that have been reported in the software engineering literature. In addition, an analysis of groups of related faults provided additional errors types that were included in the requirement error taxonomy.
- A description of human errors (from cognitive psychology) and their classifications accounted for additional errors that can occur during the requirement stage.
- Finally, a requirement error taxonomy that classifies all of the errors uncovered during the systematic review along with their impact on software quality using example of faults they are likely to cause by some of the errors.

6.2. Strengths and weaknesses

Validity of evidence: The evidence collected from the literature was identified through a search of multiple literature databases covering all of the relevant journals, proceedings, technical reports and other literature in this area. To reduce bias, data extraction forms were utilized to consistently extract the desired information from each of the selected papers. The information extracted was validated through comparison of independent analysis results between the authors of the paper. Finally, a well-defined inclusion and exclusion criterion was followed to select only the most appropriate papers.

Results of quality assessment: The results from the quality assessment (Table 5) showed that the papers were of high quality. The papers that address research question 1 were all experiments that satisfied the four quality questions, i.e., they provided empirical evidence on the use of errors for improving software quality, described the analysis method appropriately and were completely understandable.

The papers collected that address research question 2 were a mixture of experiments and observational studies. These studies analyzed the root causes of faults in the software requirement phase and other software lifecycle phases. All the studies provided error information using well-described analyzes, used real defect data from software organizations, and presented the evidence (e.g., by describing case studies of using error information to improve software process or decreasing defect rates) or observations (e.g., by noticing better customer satisfaction and improved communication) to support their findings and conclusions.

The papers that address research question 3 were also a mixture of experiments and observational studies. The observational studies described human error classifications derived from observing the human thought processes while planning, making decisions, and problem solving. The experiments described new human error classification and validated those described in the observational studies. The human errors described in these studies were supported by evidence (empirical or observations).

Adequacy ofrequirement error taxonomy to address limitations: To overcome the limitations and drawbacks identified in Section 2 (listed in Table 1) and discussed in Section 4 (in the answer to *question 1.2*), the requirement error taxonomy provides comprehensive error information to help developers identify and classify errors in the requirements phase. The usefulness of the requirement error taxonomy has been validated through a series of three controlled experiments at Mississippi State University. The initial results from these studies have been published, with more detailed analysis ongoing. Therefore, only an overview of how the requirement error taxonomy can be used to improve the quality of software artifacts and the major results of each study are provided.

To evaluate the feasibility of using a requirement error taxonomy, an initial taxonomy was developed using an *ad hoc* review (as opposed to this systematic review). Then, a study was conducted to ensure that developers could use the error information to improve their defect detection capability during the inspection process. This study utilized a repeated-measures experiment design, where the subjects were asked to first inspect the requirement artifact (which they created) to find as many faults as possible. These subjects were then trained in the use of error abstraction (similar to Lanubile et al. [53]) following by training in the use of error classification (using the requirement error taxonomy). The subjects were then asked to return to artifact and re-inspect it using the knowledge of errors that they committed to identify additional faults. The results from this study showed that all the subjects found a considerably large number of additional faults during the second inspections [89,90].

After this study, the systematic review reported in this paper was conducted to develop the requirement error taxonomy as described in Section 5. A second study was then performed using the new taxonomy. This study replicated the first study with the addition of a control group. The goal of this study was to investigate

how much of the increase seen during the re-inspection step of study 1 was due to the requirement error taxonomy and how much was due simply to performing a second inspection of the same artifact. In this study, the control group simply performed two inspections to look for faults, while the treatment group performed the first inspection in the same way as control group followed by error abstraction and classification to inform their second inspection (as in study 1). The results from this study showed that the experiment group, using the requirement error taxonomy, was significantly more effective than the control group during the re-inspection [88,90].

A third study also evaluated the usefulness of the requirement error taxonomy by replicating the first study and adding an observational variable. The goal of this study was to observe firsthand the behavior and interactions of subjects while they developed the requirements document and inspected it using the requirement error taxonomy. This study included a participant observer (one of the researchers) who took notes about the errors committed by subjects during the development of the requirements and about their interactions when using the requirement error taxonomy. The results from this study also confirmed the effectiveness of using the requirement error taxonomy to ensure early defect detection.

In addition, the subjects participating in all the three studies favored the requirement error taxonomy on different attributes (i.e., simplicity, understandability, applicability, intuitiveness, orthogonal, comprehensiveness, usefulness, and uniformity across products). The results from all the three studies showed that the requirement error taxonomy: (a) improves the effectiveness for inspection teams and individual inspectors, (b) is useful for improving the quality of a requirements document, and (c) provides important insights into the requirement phase of the software lifecycle. The results also support the validity of using human cognition research to better understand software fault production, and to increase defect detection [88–90]. These studies with university students have provided positive initial results and further validation is underway.

6.3. Contribution to research and practice communities

This paper provides a new perspective for the investigation of software quality improvement. It outlines the software quality problem and describes the limitations of the existing practices in ensuring software quality. A systematic review encompassing a vast body of literature from both the software engineering and cognitive psychology fields was conducted to identify and classify requirements errors. The result of the review is a requirement error taxonomy to help developers identify errors at their origin and ensure the early detection of defects. Based on the initial empirical evaluations, the requirement error taxonomy shows promise as an effective quality improvement approach and provides important insights into the requirement phase of software lifecycle.

This work will contribute to the understanding of software quality assurance through the inclusion of research from cognitive psychology. Researchers and practitioners from both domains will benefit from this work. For researchers, this work can serve as a starting point for future research into the development of similar errors taxonomies for other lifecycle phases. We also anticipate that the continuing work in this area will provide additional insights into the cognitive aspects of software development and quality assurance. For practitioners, software engineers will gain a more solid psychological basis for understanding software faults and errors, and developers will be provided with a new set of tools to prevent, detect, and repair the errors and resulting faults. Future research into the development of concrete techniques based on the error taxonomies can be used by developers to efficiently identify the cause of faults at their source, and improve the quality of their software.

6.4. Conclusion and future work

Based on the evidence gathered from the review and the data collected from initial studies, the requirement error taxonomy shows promise. The evidence provided in this paper has motivated further investigation of the effectiveness of the proposed taxonomy. These results provide additional information for researchers who are investigating methods to improve quality by developing more effective tools and methods to assist software developers.

Finally, similar systematic reviews to identify errors in the subsequent software lifecycle phases are needed. Research in this direction will help the research and practice communities develop a more complete verification process.

Acknowledgements

We thank the Empirical Software Engineering (ESE) Research Group at Mississippi State University for providing useful feedback and suggestions during the systematic review. We also thank Dr.

Table 28
Search strings.

String #	High level search string	Detailed search string	Review question
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))	1
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))	2
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
7	Contribution from other fields to S/W development	((contribution OR significance OR benefit OR supplement OR assistance) **AND** (human cognition OR psychology) **AND** (software development))	

Table 29
List of requirement errors identified in software engineering literature.

Description of error	Reference
Methods (which may be incomplete, ambiguous, wrong, or unenforced); tools and environment (which may be clumsy, unreliable, or defective); and people (who may lack adequate training and understanding)	[24]
Poor organization of requirements	[41]
No use of standard format used for documenting requirements	
Inadequate/insufficient training and experience	
Inadequate project communications	
Changes in the requirements not communicated	[46]
Not involvement of all the stakeholders	
Poor impact analysis of changing requirements	
Lack of trust	
People have different interpretation of requirements	
Communication error (lack of communication among developers and between developer and users); transcription error (the developer understood everything but simply made a mistake); and the developer did not understand some aspect of the product or process	[60]
User needs not well-understood or interpreted by different stakeholders	[61]
Missing checks (item exists but forgotten)	
Mishandling of steps to follow	
Lack of understanding of the system	
Lack of domain knowledge or lack of system knowledge	[55]
Communication problems	
Individual mistakes	
Lack of change coordination	
Lack of domain knowledge	[8]
Misunderstanding of problem solution processes	
Mistakes in developing models for analyzing requirements	
Ineffective method of organizing individual requirements	
Carelessness while documenting requirements	
Mishandling of certain processes	
Poor communication between users and developers, and between members of the development teams	[47]
Lack of involvement of users at all times during requirement development	
Lack of communication between sub teams; and inadequate requirement development processes	[16]
Communication errors between development teams	[13]
Lack of user communication	
Complexity of problem domain	
Problem in assignment of resources to different tasks	
Lack of proper methods for collecting requirements	
Inadequate requirement traceability	
Problem while analyzing the solution space	
Communication issues between users and developers	[21]
Complexity of problem domain	
Constraints on humans as information processors	
Lack of developer communication and lack of user communication	[42]
Complexity of the application domain	
Inadequate requirement traceability	
Undefined requirement process	
Lack of skills or inappropriate skills or lack of training	
Inadequate assignment of resources	
Poor communication and interactions among users and developers thorough the requirement development process	[81]
Lack of trained or experienced personnel	
Misunderstandings of requirements by development team	
Lack of domain knowledge or lack of specific task knowledge	[82]
Clerical errors	[7]
Mistaken assumptions about the problem space	
Unclear lines of communication and authority	[39]
Lack of involvement due to internal factors like rivalry among developers or lack of the motivation	
Only relying on selected users to accurately define all the requirements	
Different technical standards followed by sub-teams	
Lack of proper environment	
Lack of experience and poor communication among developers	
Lack of proper methods of collecting requirements	[76]
Inadequate setting of goals and objectives	
Complex domain	[12]
Lack of domain knowledge	
Lack of communication	
Poor communication among developers	[27]
Lack of appropriate knowledge about the application and lack of awareness of sources of requirements; and poor management of people and resources	
Lack of domain knowledge	[92]
Human nature (mistakes or omissions)	
Lack of management leadership and motivation	[29]
Lack of skills and poor planning	
Impact analysis not systematically achieved and change requests insufficiently formalized	
Communication problems	[33]
Simple omission	
Wrong solution chosen because some system specific information was misunderstood	
Not understanding some parts of the problem domain	[65]

(*continued on next page*)

Table 29 (*continued*)

Description of error	Reference
Lack of domain knowledge	[40]
Lack of task knowledge	
Mistakes in setting goals	
Incomplete knowledge leading to poor plans on achieving goals	
Complexity of the task	
Application of wrong rules	
Lack of involvement of all the users	
Lack of communication within a team or between teams	[56]
Misunderstanding of interface specifications	
Error in selecting a choice of solution	
Mistaken assumptions	
Lack of communication of the requirement changes	
Misunderstandings about the timing constraints, data dependency constraints, and command ordering constraints	[57]
Lack of motivation, inadequate communications, and poor planning	[87]
Lack of communication among group of people working together	[71]
Lack of knowledge, skills, or experience to perform a task	
Deficiency of resource or task management	

Gary Bradshaw, a human cognition and psychology expert, for providing information on relevant databases and useful feedback on our research. We thank Doris Carver and Guilherme Travassos for reviewing early drafts of this paper. Finally we thank the anonymous reviewers for their helpful comments.

Appendix A

To search these databases described in Table 3, a set of search strings was created for each research question based on keywords extracted from the research questions and augmented with synonyms. Table 12 shows the search strings used for each research question (three search strings for question 1, two search strings for question 2, and two search strings for question 3). In developing the keyword strings, the following principles were applied:

- The major terms were extracted from the review questions and augmented with other terms known to be relevant to the research.
- A list of meaningful synonyms, abbreviations, and alternate spellings was then generated. This list also included terms identified via consultation with the librarian and additional terms from papers that were known to be relevant.
- The following global search string was constructed containing all of the relevant keywords and their synonyms.

((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)* ***AND*** *(error* ***OR*** *mistake* ***OR*** *problem* ***OR*** *reason* ***OR*** *fault* ***OR*** *defect* ***OR*** *imperfection* ***OR*** *flaw* ***OR*** *lapse* ***OR*** *slip* ***OR*** *err)* ***AND*** *(requirement* ***OR*** *specification)* ***AND*** *(phase* ***OR*** *stage* ***OR*** *situation* ***OR*** *division* ***OR*** *period* ***OR*** *episode* ***OR*** *part* ***OR*** *state* ***OR*** *facet)* ***AND*** *(taxonomy* ***OR*** *classification* ***OR*** *categorization* ***OR*** *grouping* ***OR*** *organization* ***OR*** *terminology* ***OR*** *systematization)* ***AND*** *(human* ***OR*** *cognitive* ***OR*** *individual* ***OR*** *psychological)* ***AND*** *(abstraction* ***OR*** *root cause* ***OR*** *cause)* ***AND*** *(inspection* ***OR*** *assessment* ***OR*** *evaluation* ***OR*** *examination* ***OR*** *review* ***OR*** *measurement)* ***AND*** *(contribution* ***OR*** *significance* ***OR*** *assistance* ***OR*** *benefit* ***OR*** *supplement)* ***AND*** *(human cognition* ***OR*** *psychology* ***OR*** *failure management)).*

Using this global search string, seven different search strings (each one with its own purpose) were derived and executed on each database. These strings are explained in Table 28 with the following attributes: *string #* – identification; *high level search string* – the high level set of keywords used to derive the string; *detailed search string* – the more detailed set of keywords used to derive the string (this string was customized for each database search options to generate best results); and *review question* – maps to the primary research questions.

Appendix B

This appendix describes the all the requirement errors that were identified in the software engineering literature. The sources used to identify requirement errors are shown in Table 10 of the journal paper under research question 2.1, and described in Table 29.

Table 30 describes the list of other software process errors (in design and coding phases), and are related to errors that can occur during the requirement stage. Table 30 is related to the research question 2.2 of the journal paper.

Table 31 describes the list of requirement errors that were abstracted from analysis of source of actual faults. Table 31 is related to the research question 2.3 of the journal paper.

Table 30
List of other software process errors that can also occur during the requirement.

Description of error	Reference
Miscommunication among teams	[17,41]
Lack of domain or system knowledge	
Misunderstandings caused by working simultaneously with several different software systems and domains	
Using an analogy to derive a sequence of actions from other similar situations resulting in the wrong choice of a sequence of actions	[63,64]
Mistake while executing the sequence of actions	
Misunderstanding of situation while forming a goal	
Technological, historical, organizational, individual or other causes	[31]
Inattention or over attention	[50,51]
Choosing the wrong plan due to information overload	
Wrong actions to achieve goals	
Incorrect model while trying to understand solution and task complexity	

Table 31
List of abstracted errors.

Description of error
Lack of participation of all the stakeholders
Mistakes or misunderstandings in mapping inputs, outputs, and processes
Unresolved system interfaces and unanticipated dependencies
Misunderstanding of some aspect of the overall functionality of the system
Mistakes while analyzing requirement use cases or different scenarios in which the system can be used

Appendix C

This appendix provides a list of all the papers that provided input to this review, including those cited in the main paper and some additional references. The additional references were not included in the main paper because they are either illustration of error taxonomies that were cited in the main paper, or they are references that do not add any new information to those that are cited. This complete list of references is included for the sake of completeness and may be of interest to some readers. For the sake of clarity, references marked with a '*' are included in the reference list for the main paper.

*[1] IEEE Std 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology, 1990.

*[2] Software Engineering Laboratory: Software Measurement Guidebook, NASA/GSFC Software Engineering Laboratory, 1994.

*[3] A.F. Ackerman, L.S. Buchwald, F.H. Lewski, Software inspections: an effective verification process, IEEE Software 6 (3) (1989) 31–36.

*[4] A. Aurum, H. Petersson, C. Wohlin, State-of-the-art: software inspections after 25 years, Journal of Software Testing Verification and Reliability 12 (3) (2002) 133–154.

[5] B.P. Bailey, J.A. Konstan, On the need for attention-aware systems, measuring effects of interruption on task performance, error rate and affective state. Journal of Computers in Human Behavior 22 (4) (2006) 685–708.

[6] V. Basili, S. Green, Software process evolution at the SEL, IEEE Software 11 (2) (1994) 58–66.

*[7] V.R. Basili, D.M. Weiss, Evaluation of a software requirements document by analysis of change data, in: Proceedings of the Fifth International Conference on Software Engineering, IEEE Press, San Diego, CA, 1981, pp. 314–323.

*[8] V.R. Basili, B.T. Perricone, Software errors and complexity: an empirical investigation, Communications of the ACM 27 (1) (1984) 42–52.

[9] V.R. Basili, E.E. Katz, N.N. Panlilio-Yap, C.L. Ramsey, S. Chang, Characterization of an ada software development, IEEE Computer 18 (9) (1985).

*[10] V.R. Basili, H.D. Rombach, Tailoring the software process to project goals and environments, in: Proceedings of Ninth International Conference in Software Engineering, IEEE Press, California, United States, 1987, pp. 345–357.

*[11] V.R. Basili, H.D. Rombach, The TAME project: towards improvement-oriented software environments, IEEE Transactions on Software Engineering 14 (6) (1988) 758–772.

*[12] V.R. Basili, Evolving and packaging reading techniques, Journal of Systems and Software 38 (1) (1997) 3–12.

*[13] S. Basu, N. Ebrahimi, Estimating the number of undetected errors: Bayesian model selection, in: Proceedings of the Ninth International Symposium on Software Reliability Engineering, IEEE Computer Society, Paderborn, Germany, 1998, pp. 22–31.

*[14] D. Batra, Cognitive complexity in data modeling: causes and recommendations, Requirements Engineering Journal 12 (4) (2007) 231–244.

*[15] S. Beecham, T. Hall, C. Britton, M. Cottee, A. Rainer, Using an expert panel to validate a requirements process improvement model, The Journal of Systems and Software 76 (3) (2005) 251–275.

*[16] T.E. Bell, T.A. Thayer, Software requirements: are they really a problem? in: Proceedings of Second International Conference on Software Engineering, IEEE Computer Society Press, Los Alamitos, CA, 1976, pp. 61–68.

[17] J.P. Benson, S.H. Saib, A software quality assurance experiment, in: Proceedings of the Software Quality Assurance Workshop on Functional and Performance Issues, ACM Press, 1978, pp. 87–91.

*[18] T. Berling, T. Thelin, A case study of reading techniques in a software company, in: Proceedings of the 2004 International Symposium on Empirical Software Engineering (ISESE'04), IEEE Computer Society, 2004, pp. 229–238.

[19] B. Bernardez, M. Genero, A. Duran, M. Toro, A controlled experiment for evaluating a metric-based reading technique for requirement inspection, in: Proceedings of the 10th International Symposium on Software Metrics (METRICS'04), IEEE Computer Society, 2004, pp. 257–268.

*[20] I. Bhandari, M. Halliday, E. Tarver, D. Brown, J. Chaar, R. Chillarege, A case study of software process improvement during development, IEEE Transactions on Software Engineering 19 (12) (1993) 1157–1170.

*[21] I. Bhandari, M.J. Halliday, J. Chaar, R. Chillarege, K. Jones, J.S. Atkinson, C. Lepori-Costello, P.Y. Jasper, E.D. Tarver, C.C. Lewis, M. Yonezawa, In process improvement through defect data interpretation, IBM Systems Journal 33 (1) (1994) 182–214.

*[22] J. Biolchini, P.G. Mian, A.C. Natatli, G.H. Travassos, Systematic Review in Software Engineering: Relevance and Utility, PESC-COPPE/UFRJ, Brazil, 2005, <http://cronos.cos.ufrj.br/publicacoes/reltec/es67905.pdf>.

[23] S. Biffl, B. Freimut, O. Laitenberger, Investigating the cost-effectiveness of reinspections in software development, in: Proceedings of the 23rd International Conference on Software Engineering, IEEE Computer Society, Toronto, Ontario, Canada, 2001, pp. 155–164.

[24] M.R. Blackburn, R. Busser, A. Nauman, Eliminating requirement defects and automating test, in: Test Computer Software Conference, IEEE Computer Society, 2001, pp. 25–34.

[25] A. Blavier, E. Rouy, A.S. Nyssen, V. Keyser, Prospective issues for error detection, Journal of Ergonomics 48 (7) (2005) 758–781.

*[26] B. Boehm, V.R. Basili, Software defect reduction top 10 list, IEEE Computer 34 (1) (2001) 135–137.

*[27] T. Bove, Development and Validation of Human Error Management Taxonomy in Air Traffic Control, Risø National Laboratory and University of Roskilde, 2002, p. 234.

*[28] G.J. Browne, V. Ramesh, Improving information requirements determination: a cognitive perspective, Journal of Information and Management 39 (8) (2002) 625–645.

*[29] B. Brykczynski, A survey of software inspection checklists, ACM SIGSOFT Software Engineering Notes 24 (1) (1999) 82–89.

[30] D.K. Buse, C.W. Johnson, Identification and analysis of incidents in complex, medical environments, in: Proceedings of the First Workshop on Human Error and Clinical Systems, Cambridge University Press, Glasgow, Scotland, 1999.

*[31] P.C. Cacciabue, A methodology of human factors analysis for systems engineering: theory and applications, IEEE Transactions on System, Man and Cybernetics – Part A: Systems and Humans 27 (3) (1997) 325–329.

*[32] D.N. Card, Learning from our mistakes with defect causal analysis, IEEE Software 15 (1) (1998) 56–63.

*[33] B. Cheng, R. Jeffrey, Comparing inspection strategies for software requirement inspections, in: Proceedings of the 1996 Australian Software Engineering Conference, IEEE Computer Society, Melbourne, Australia, 1996, pp. 203–211.

[34] Y. Chernak, A statistical approach to the inspection checklist formal synthesis and improvement, IEEE Transactions on Software Engineering 22 (12) (1996) 866–874.

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

[36] M. Ciolkowski, O. Laitenberger, S. Biffl, Software reviews: the state of the practice, IEEE Software 20 (6) (2003) 46–51.

[37] B. Clark, D. Zubrow, How good is the software: a review of defect prediction techniques, in: Software Engineering Symposium, IEEE Computer Press, Pittsburgh, PA, 2001, pp. 1–35.

*[38] J. Coughlan, D.R. Macredie, Effective communication in requirements elicitation: a comparison of methodologies, Requirements Engineering Journal 7 (2) (2002) 47–60.

*[39] W.J. Cristopher, The cost of errors in software development: evidence from industry, The Journal of System and Software 62 (1) (2002) 1–9.

*[40] C. Debou, A.K. Combelles, Linking software process improvement to business strategies: experiences from industry, Journal of Software Process: Improvement and Practice 5 (1) (2000) 55–64.

[41] S.W.A. Dekker, Illusions of explanation : a critical essay on error classification, The International Journal of Aviation Psychology 13 (2) (2003) 95–106.

[42] B.S. Dhillon, Y. Liu, Human error in maintenance: a review, Journal of Quality in Maintenance Engineering 12 (1) (2006) 21–36.

[43] I.M.M. Duncan, D.J. Robson, An exploratory study of common coding faults in C programs, C Traps and Pittfalls 8 (4) (1996) 251–256.

[44] K. Emam, O. Laitenberger, T. Harbich, The application of subjective estimates of effectiveness to controlling software inspections, The Journal of Systems and Software 54 (2) (2000) 119–136.

*[45] A. Endres, An analysis of errors and their causes in system programs, IEEE Transactions on Software Engineering 1 (2) (1975) 140–149.

*[46] A. Endres, D. Rombach, A Handbook of Software and Systems Engineering, first ed., Pearson Addison Wesley, Harlow, England, 2003.

*[47] M.R. Endsley, Situation awareness and human error: designing to support human performance, in: Proceedings of the High Consequence Systems Surety Conference, SA Technologies, Albuquerque, NM, 1999, pp. 2–9.

[48] M.E. Fagan, Design and code inspections to reduce errors in program development, IBM Systems Journal 15 (3) (1976) 182–211.

*[49] N.E. Fenton, M. Neil, A critique of software defect prediction models, IEEE Transactions on Software Engineering 25 (5) (1999) 675–689.

*[50] P.M. Fitts, R.E. Jones, Analysis of factors contributing to 460 'pilot error' experiences in operating aircrafts control, in: Selected Papers on Human Factors in the Design and Use of Control Systems, Dover Publications Inc., New York, 1961, pp. 332–358.

*[51] W.A. Florac, Software Quality Measurement: A Framework for Counting Problems and Defects, Carnegie Mellon Software Engineering Institute, Pittsburgh, PA, 1992.

[52] M. Fredericks, V. Basili, Using Defect Tracking and Analysis to Improve Software Quality, The Data and Analysis Center for Software (DACS), 1998.

*[53] B. Freimut, C. Denger, M. Ketterer, An industrial case study of implementing and validating defect classification for process improvement and quality management, in: Proceedings of the 11th IEEE International Software Metrics Symposium, IEEE Press, 2005.

[54] M.S. Fujii, A comparison of software assurance methods, ACM SIGMETRICS Performance Evaluation Review 7 (3–4) (1978) 27–32.

*[55] P. Fusaro, F. Lanubile, G. Visaggio, A replicated experiment to assess requirements inspection techniques, Journal of Empirical Software Engineering 2 (1) (1997) 39–57.

*[56] D.A. Gaitros, Common errors in large software development projects, The Journal of Defense Software Engineering 12 (6) (2004) 21–25.

*[57] J. Galliers, S. Minocha, A. Sutcliffe, A causal model of human error for safety critical user interface design, ACM Transactions on Computer–Human Interaction 5 (3) (1998) 756–769.

[58] J. Galliers, A. Sutcliffe, S. Minocha, An impact analysis method for safety-critical user interface design, ACM Transactions on Computer–Human Interaction 6 (4) (1999) 341–369.

[59] M. Graboswki, K.H. Roberts, Human and organizational error in large scale systems, IEEE Transactions on System, Man And Cybernetics – Part A: Systems and Humans 26 (1) (1996) 2–16.

*[60] R.B. Grady, Software failure analysis for high-return process improvement, Hewlett–Packard Journal 47 (4) (1996) 15–24.

[61] W.D. Gray, The nature and processing of errors in interactive behavior, Journal of Cognitive Science 24 (2) (2000) 205–248.

*[62] T. Hall, S. Beecham, A. Rainer, Requirement problems in twelve software companies: an empirical analysis, IEE Proceedings Software 149 (5) (2002) 153–160.

*[63] J.H. Hayes, Building a requirement fault taxonomy: experiences from a NASA verification and validation research project, in: Proceedings of the 14th International Symposium on Software Reliability Engineering, IEEE Computer Society, 2003, pp. 49–59

*[64] J.H. Hayes, E.A. Holbrook, I. Raphael, D.M. Pruett, Fault-based analysis: how history can help improve performance and dependability requirements for high assurance systems, in: Fifth International Workshop on Requirements for High Assurance Systems (RHAS'05), IEEE Computer Society, Chicago, 2005.

[65] K. Henningsson, C. Wohlin, Assuring fault classification agreement – an empirical evaluation, in: Proceedings of the 2004 International Symposium on Empirical Software Engineering (ISESE'04), IEEE Computer Society, 2004, pp. 95–104.

[66] A. Hobbs, A. Williamson, Skills, rules and knowledge in aircraft maintenance: errors in context, Journal of Ergonomics 45 (4) (2002) 290–308.

[67] E. Hollnagel, Human Reliability Analysis: Context and Control, Academic Press, London, 1994.

[68] J.C. Huang, C.K. Chang, M. Christensen, Event-based traceability for managing evolutionary change, IEEE Transactions on Software Engineering 29 (9) (2003) 796–804.
[69] W.S. Humphrey, Using a defined and measured personal software process, IEEE Software 13 (3) (1996) 77–88.
*[70] A. Issac, S.T. Shorrock, R. Kennedy, B. Kirwan, H. Andersen, T. Bove, The Human Error in ATM Technique (HERA-JANUS), European Air Traffic Management, 2002, pp. 1–94.
*[71] J. Jacobs, J.V. Moll, P. Krause, R. Kusters, J. Trienekens, A. Brombacher, Exploring defect causes in products developed by virtual teams, Journal of Information and Software Technology 47 (6) (2005) 399–410.
[72] M.S. Jaffe, N.G. Leveson, M.P.E. Heimdahl, B.E. Melhart, Software requirements analysis for real-time process control systems, IEEE Transactions on Software Engineering 17 (3) (1991) 241–258.
[73] C. Johnson, Forensic software engineering: are software failures symptomatic of software problems? Journal of Safety Science 40 (9) (2002) 835–847.
[74] M. Jorgensen, M. Shepperd, A systematic review of software development cost estimation studies, IEEE Transactions on Software Engineering 33 (1) (2007) 33–53.
[75] W. Joung, B. Hesketh, Using "war stories" to train for adaptive performance: is it better to learn from errors or success? Journal of Applied Psychology: An International Review 55 (2) (2006) 282–302.
*[76] S.H. Kan, V.R. Basili, L.N. Shapiro, Software quality: an overview from the perspective of total quality management, IBM Systems Journal 33 (1) (1994) 4–19.
[77] E. Kantorowitz, L. Arzi, A. Gutmann, The performance of the N-fold requirement inspection method, Requirement Engineering Journal 2 (3) (1997) 152–164.
[78] D. Kelly, T. Shepard, Task-directed software inspection technique: an experiment and case study, in: Proceedings of the 2000 Conference of the Centre for Advanced Studies on Collaborative Research, IBM Press, Mississauga, Ontario, Canada, 2000.
[79] T.G. Kirner, J.C. Abib, Inspection of software requirements specification documents: a pilot study, in: Proceedings of the 15th Annual International Conference on Computer Documentation, IEEE Press, Salt Lake City, Utah, United States, 1997, pp. 161–171.
[80] J.C. Knight, A.E. Myers, An improved inspection technique, Communications of the ACM 36 (11) (1993) 50–69.
*[81] A.J. Ko, B.A. Myers, Development and evaluation of a model of programming errors, in: Proceedings of IEEE Symposium on Human Centric Computing Languages and Environments, IEEE Computer Society, 2003, pp. 7–14.
*[82] A.J. Ko, B.A. Myers, A framework and methodology for studying the causes of software errors in programming systems, Journal of Visual Languages and Computing 16 (2) (2005) 41–84.
*[83] J. Kramer, A.L. Wolf, in: Succeedings of the Eighth International Workshop on Software Specification and Design, ACM SIGSOFT Software Engineering Notes, vol. 21, no. 5, 1996, pp. 21–35.
[84] J. Krogstie, Integrating the understanding of quality in requirements specification and conceptual modeling, ACM SIGSOFT Software Engineering Notes 23 (1) (1998) 86–91.
[85] O. Laitenberger, C. Atkinson, Generalizing perspective-based inspection to handle object-oriented development artifacts, in: International Conference on Software Engineering, IEEE Computer Society Press, Los Angeles, CA, USA, 1999, pp. 494–503.
[86] O. Laitenberger, C. Atkinson, M. Schlich, K.E. Emam, An experimental comparison of reading techniques for defect detection in UML design documents, The Journal of Systems and Software 53 (2) (2000) 183–204.
[87] O. Laitenberger, J.M. DeBaud, An encompassing lifecycle centric survey of software inspection, The Journal of Systems and Software 50 (1) (2000) 5–31.
[88] F. Lanubile, G. Visaggio, Assessing Defect Detection Methods for Software Requirement Inspection Through External Replication, Department of Informatica, University of Bari, 1996.
*[89] F. Lanubile, F. Shull, V.R. Basili, Experimenting with error abstraction in requirements documents, in: Proceedings of the Fifth International Symposium on Software Metrics (METRIC'98), IEEE Computer Society, Bethesda, MD, USA, 1998, pp. 114–121.
*[90] C.P. Lawrence, I. Kosuke, Design error classification and knowledge management, Journal of Knowledge Management Practice 10 (9) (2004) 72–81.
*[91] M. Leszak, D.E. Perry, D. Stoll, A case study in root cause defect analysis, in: Proceedings of the 22nd International Conference on Software Engineering, ACM Press, Limerick, Ireland, 2000, pp. 428–437.
[92] R.R. Lutz, K. Wong, S. Johnny, Constraint checking during error recovery, in: Proceedings of the NASA Technology 2002 Conference, IEEE Computer Society, 1992, pp. 142–153.
*[93] R.R. Lutz, Analyzing software requirements errors in safety-critical, embedded systems, in: Proceedings of the IEEE International Symposium on Requirements Engineering, IEEE Computer Society Press, San Diego, CA, USA, 1993, pp. 126–133.
*[94] R.R. Lutz, Targeting safety-related errors during software requirements analysis, The Journal of Systems and Software 34 (3) (1996) 223–230.
[95] R.R. Lutz, Requirements analysis using forward and backward search, Annals of Software Engineering 3 (1) (1997) 459–475.
*[96] J. Martin, W.T. Tsai, N-fold inspection: a requirement analysis technique, Communications of the ACM 33 (2) (1990) 225–232.
*[97] C. Masuck, Incorporating a fault categorization and analysis process in the software build cycle, Journal of Computing Sciences in Colleges 20 (5) (2005) 239–248.
*[98] R.G. Mays, C.L. Jones, G.J. Holloway, D.P. Studinski, Experiences with defect prevention, IBM Systems Journal 29 (1) (1990) 4–32.
[99] F. McGarry, G. Page, V. Basili, M. Zelkowitz, Software Process Improvement in the NASA Software Engineering Laboratory, Carnegie Mellon Software Engineering Institute (SEI), 1994.
[100] K.S. Mendis, Quantifying software quality, Annual Quality Congress Transaction 35 (4) (1981) 11–18.
[101] J. Miller, M. Wood, M. Roper, Further experiences with scenarios and checklists, Journal of Empirical Software Engineering 3 (1) (1998) 37–64.
[102] Y. Mohri, T. Kikuno, Fault analysis based on fault reporting in JSP software development, in: Proceedings of the 15th Annual International Computer Software and Applications Conference, IEEE Computer Society, 1991, pp. 591–596.
[103] J.C. Munson, A.P. Nikora, Toward a quantifiable definition of software faults, in: Proceedings of the 13th International Symposium on Software Reliability Engineering, IEEE Computer Society, 2002, pp. 388–395.

[104] N. Nachiappan, W. Laurie, H. John, S. Will, V. Mladen, Preliminary results on using static analysis tools for software inspection, in: Proceedings of the 15th International Symposium on Software Reliability Engineering (ISSRE'04), IEEE Press, 2004, 429–439.
*[105] T. Nakashima, M. Oyama, H. Hisada, N. Ishii, Analysis of software bug causes and its prevention, Journal of Information and Software Technology 41 (15) (1999) 1059–1068.
[106] C.R. Nelms, The latent causes of industrial failures how to identify them, and what to do about them, in: Proceedings of the IEEE Sixth Conference on Human Factors and Power Plants, 1997, Global Perspectives of Human Factors in Power Generation, IEEE Press, Orlando, FL, USA, 1997, pp. 7–12.
[107] J.M. Nieves, A.P. Sage, Human and organizational error as a basis for process reengineering: with applications to system integration planning and marketing, IEEE Transactions on System, Man And Cybernetics – Part A: Systems and Humans 28 (6) (1998) 742–762.
*[108] D. Norman, The Psychology of Every Day Things, Basic Books, New York, 1988.
*[109] D.A. Norman, Steps towards a cognitive engineering: design rules based on analyzes of human error, Communications of the ACM 26 (4) (1981) 254–258.
*[110] D.A. Norman, Design rules based on analyzes of human error, Communications of the ACM 26 (4) (1983) 254–258.
*[111] K.M. Oliveira, F. Zlot, A.R. Rocha, G.H. Travassos, C. Galotta, C.S. Menezes, Domain-oriented software development environment, The Journal of Systems and Software 72 (2) (2004) 145–161.
*[112] F. Paterno, C. Santoro, Preventing user errors by systematic analysis of deviations from the system task model, International Journal of Human–Computer Studies 56 (2) (2002) 225–245.
[113] F. Patrick, L. David, M. Melinda, P. Andy, Tree-based methods for classifying software failures, in: Proceedings of the 15th International Symposium on Software Reliability Engineering (ISSRE'04), IEEE Press, 2004, pp. 451–462.
*[114] A.A. Porter, L.G. Votta, V.R. Basili, Comparing detection methods for software requirements inspections: a replicated experiment, IEEE Transactions on Software Engineering 21 (6) (1995) 563–575.
[115] A.A. Porter, L.G. Votta, What makes inspections work? IEEE Software 14 (6) (1997) 99–102.
[116] L. Prechelt, Accelerating learning from experience: avoiding defects faster, IEEE Software 18 (6) (2001) 56–61.
*[117] J. Reason, Human Error, Cambridge University Press, Cambridge, USA, 1990.
[118] S. Reinach, A. Viale, Application of a human error framework to conduct train accident/incident investigations, Journal of Accident Analysis and Prevention 38 (2) (2006) 396–406.
[119] P.N. Robillard, The role of knowledge in software development, Communications of the ACM 42 (1) (1999) 87–92.
[120] W.N. Robinson, S.D. Pawlowski, Managing requirement inconsistency with development goal monitors, IEEE Transactions on Software Engineering 25 (6) (1999) 816–835.
[121] G. Sabaliauskaite, S. Kusumoto, K. Inoue, Assessing defect detection performance of interacting teams in object-oriented design inspections, Journal of Information and Software Technology 46 (13) (2004) 875–886.
*[122] S. Sakthivel, A survey of requirement verification techniques, Journal of Information Technology 6 (2) (1991) 68–79.
[123] K. Sandahl, O. Blomkvist, J. Karlsson, C. Krysander, M. Lindvall, N. Ohlsson, An extended replication of an experiment for assessing methods for software requirements inspections, Journal of Empirical Software Engineering 3 (4) (1998) 327–354.
*[124] K. Sasao, J. Reason, Team errors: definition and taxonomy, Journal of Reliability Engineering and System Safety 65 (1) (1999) 1–9.
[125] P. Sawyer, I. Sommerville, S. Viller, Capturing the benefits of requirement engineering, IEEE Software 16 (2) (1999) 78–85.
*[126] G.M. Schneider, J. Martin, W.T. Tsai, An experimental study of fault detection in user requirements documents, ACM Transactions on Software Engineering and Methodology 1 (2) (1992) 188–204.
*[127] L.W. Senders, N.P. Moray, Human Error: Cause, Prediction, and Reduction, Lawrence Erlbaum, Hillslade, NJ, 1991.
[128] S.A. Shappell, D.A. Weigmann, A human error approach to accident investigation: the taxonomy of unsafe operations, The International Journal of Aviation Psychology 7 (4) (1997) 269–291.
[129] W. Shen, M. Guizani, Z. Yang, K. Compton, J. Huggins, Execution of a requirement model in software development, in: Proceedings of the 13th International Conference on Intelligent and Adaptive Systems and Software Engineering, IEEE Press, Nice, France, 2004, pp. 203–208.
*[130] S.T. Shorrock, B. Kirwan, Development and application of a human error identification tool for air traffic control, Journal of Applied Ergonomics 33 (4) (2002) 319–336.
*[131] F. Shull, I. Rus, V. Basili, How perspective based reading can improve requirement inspection, IEEE Computer 33 (7) (2000) 73–79.
[132] I. Sommerville, Software Engineering, eighth ed., Addison Wesley, Harlow, England, 2007.
*[133] J. Smith, The 40 root causes of troubled IT projects, Journal of IEEE Computer and Control Engineering 13 (3) (2002) 109–112.
*[134] N.A. Stanton, S.V. Stevenage, Learning to predict human error: issues of acceptability, reliability and validity, Journal of Ergonomics 41 (11) (1998) 1737–1756.
*[135] M.A. Stutzke, C.S. Smidts, A stochastic model of fault introduction and removal during software development, IEEE Transactions on Reliability 50 (20) (2001) 184–193.
*[136] A. Sutcliffe, G. Rugg, A taxonomy of error types for failure analysis and risk assessment, International Journal of Human–Computer Interaction 10 (4) (1998) 381–405.
*[137] A. Sutcliffe, A. Economou, P. Markis, Tracing requirements errors to problems in the requirements engineering process, Requirements Engineering Journal 4 (3) (1999) 134–151.
*[138] A. Sutcliffe, B. Gault, N. Maiden, ISRE: immersive scenario-based requirements engineering with virtual prototypes, Requirements Engineering Journal 10 (1) (2004) 95–111.
*[139] A. Swain, H. Guttman, Handbook of Human Reliability Analysis with Emphasis on Nuclear Power Plant Applications, Nuclear Regulatory Commission, Washington, DC, 1983.
*[140] T. Thelin, P. Runeson, C. Wohlin, T. Olsson, C. Andersson, Evaluation of usage based reading-conclusion after three experiments, Journal of Empirical Software Engineering 9 (1–2) (2004) 77–110.

*[141] C. Trevor, S. Jim, C. Judith, K. Brain, Human Error in Software Generation Process, University of Technology, Loughborough, England, 1994.
[142] V. Venkatasubramaniam, R. Rengaswamy, S.K. Kavuri, A review of process fault detection and diagnosis Part 2: qualitative models and search strategies, Journal of Computers and Chemical Engineering 27 (2) (2003) 313–326.
[143] V. Venkatasubramaniam, R. Rengaswamy, K. Yin, S.K. Kavuri, A review of fault detection and diagnosis Part 1: quantitative model-based methods, Journal of Computers and Chemical Engineering 27 (2) (2003) 293–311.
*[144] S. Viller, J. Bowers, T. Rodden, Human factors in requirement engineering: a survey of human sciences literature relevant to the improvement of dependable systems development processes, in: Cooperative Systems Engineering Group Technical Report, Computing Department, Lancaster University, Lancaster, 1997.
*[145] G.S. Walia, J. Carver, T. Philip, Requirement error abstraction and classification: an empirical study, in: IEEE Symposium on Empirical Software Engineering, ACM Press, Brazil, 2006, pp. 336–345.
*[146] G. Walia, J. Carver, T. Philip, Requirement error abstraction and classification: a control group replicated study, in: 18th IEEE International Symposium on Software Reliability Engineering, Trollhättan, Sweden, 2007.
*[147] G.S. Walia, J. Carver, Development of a Requirement Error Taxonomy as a Quality Improvement Approach: A Systematic Literature Review MSU-070404, Department of Computer Science and Engineering, Mississippi State University, 2007, <http://www.cse.msstate.edu/PUBLICATIONS/TECHNICAL_REPORTS/MSU-070404.pdf>.
*[148] G.S. Walia, Empirical Validation of Requirement Error Abstraction and Classification: A Multidisciplinary Approach, M.S. Thesis, Computer Science and Engineering, Mississippi, Starkville, 2006.
[149] D.M. Weiss, Evaluating software development by error analysis: the data from architecture research facility, Journal of Systems and Software 4 (4) (1979) 289–300.
[150] T. Yamamura, K. Yata, T. Yasushi, H. Yamaguchi, A basic study on human error in communication network operation, in: IEEE Global Telecommunications Conference, 1989, and Exhibition, Communications Technology for the 1990s and Beyond, GLOBECOM'89, IEEE Press, Dallas, TX, USA, 1989, pp. 795–800.
[151] D. Zage, W. Zage, An analysis of the fault correction process in a large-scale SDL production model, in: Proceedings of the 25th International Conference on Software Engineering, IEEE Computer Society, Portland, Oregon, 2003, pp. 570–577.
*[152] X. Zhang, H. Pham, An analysis of factors affecting software reliability, The Journal of Systems and Software 50 (1) (2000) 43–56.

References

[1] IEEE Std 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology, 1990.
[2] Software Engineering Laboratory: Software Measurement Guidebook, SEL-94-002, NASA/GSFC Software Engineering Laboratory, 1994.
[3] A.F. Ackerman, L.S. Buchwald, F.H. Lewski, Software inspections: an effective verification process, IEEE Software 6 (3) (1989) 31–36.
[4] E.B. Allen, Computer Aided Dispatch System for the London Ambulance Service: Software Requirement Specification, MSU-030429, Department of Computer Science and Engineering, Mississippi State University, 2003.
[5] A. Aurum, H. Petersson, C. Wohlin, State-of-the-art: software inspections after 25 years, Journal of Software Testing Verification and Reliability 12 (3) (2002) 133–154.
[6] V.R. Basili, D.M. Weiss, Evaluation of a software requirements document by analysis of change data, in: Proceedings of the Fifth International Conference on Software Engineering, IEEE Press, San Diego, CA, 1981, pp. 314–323.
[7] V.R. Basili, B.T. Perricone, Software Errors and Complexity: An Empirical Investigation, Communications of the ACM 27 (1) (1984) 42–52.
[8] V.R. Basili, H.D. Rombach, Tailoring the software process to project goals and environments, in: Proceedings of Ninth International Conference in Software Engineering, IEEE Press, California, United States, 1987, pp. 345–357.
[9] V.R. Basili, H.D. Rombach, The TAME project: towards improvement-oriented software environments, IEEE Transactions on Software Engineering 14 (6) (1988) 758–772.
[10] V.R. Basili, Evolving and packaging reading techniques, Journal of Systems and Software 38 (1) (1997) 3–12.
[11] S. Basu, N. Ebrahimi, Estimating the number of undetected errors: bayesian model selection, in: Proceedings of the Ninth International Symposium on Software Reliability Engineering, IEEE Computer Society, Paderborn, Germany, 1998, pp. 22–31.
[12] D. Batra, Cognitive complexity in data modeling: causes and recommendations, Requirements Engineering Journal 12 (4) (2007) 231–244.
[13] S. Beecham, T. Hall, C. Britton, M. Cottee, A. Rainer, Using an expert panel to validate a requirements process improvement model, The Journal of Systems and Software 76 (3) (2005) 251–275.
[14] T.E. Bell, T.A. Thayer, Software requirements: are they really a problem?, in: Proceedings of Second International Conference on Software Engineering, IEEE Computer Society Press, Los Alamitos, CA, 1976, pp 61–68.
[15] T. Berling, T. Thelin, A case study of reading techniques in a software company, in: Proceedings of the 2004 International Symposium on Empirical Software Engineering (ISESE'04), IEEE Computer Society, 2004, pp. 229–238.
[16] I. Bhandari, M. Halliday, E. Tarver, D. Brown, J. Chaar, R. Chillarege, A case study of software process improvement during development, IEEE Transactions on Software Engineering 19 (12) (1993) 1157–1170.
[17] I. Bhandari, M.J. Halliday, J. Chaar, R. Chillarege, K. Jones, J.S. Atkinson, C. Lepori-Costello, P.Y. Jasper, E.D. Tarver, C.C. Lewis, M. Yonezawa, In process improvement through defect data interpretation, IBM Systems Journal 33 (1) (1994) 182–214.
[18] J. Biolchini, P.G. Mian, A.C. Natatli, G.H. Travassos, Systematic Review in Software Engineering: Relevance and Utility, PESC-COPPE/UFRJ, Brazil, 2005, <http://cronos.cos.ufrj.br/publicacoes/reltec/es67905.pdf>.
[19] B. Boehm, V.R. Basili, Software defect reduction top 10 list, IEEE Computer 34 (1) (2001) 135–137.
[20] T. Bove, Development and Validation of Human Error Management Taxonomy in Air Traffic Control, Ph.D. Thesis, Risø National Laboratory and University of Roskilde, 2002.
[21] G.J. Browne, V. Ramesh, Improving information requirements determination: a cognitive perspective, Journal of Information and Management 39 (8) (2002) 625–645.
[22] B. Brykczynski, A survey of software inspection checklists, ACM SIGSOFT Software Engineering Notes 24 (1) (1999) 82–89.
[23] P.C. Cacciabue, A methodology of human factors analysis for systems engineering: theory and applications, IEEE Transactions on System, Man and Cybernetics – Part A: Systems and Humans 27 (3) (1997) 325–329.
[24] D.N. Card, Learning from our mistakes with defect causal analysis, IEEE Software 15 (1) (1998) 56–63.
[25] B. Cheng, R. Jeffrey, Comparing inspection strategies for software requirement inspections, in: Proceedings of the 1996 Australian Software Engineering Conference, IEEE Computer Society, Melbourne, Australia, 1996, pp. 203–211.
[26] R. Chillarege, I.S. Bhandari, J.K. Chaar, M.J. Halliday, D.S. Moebus, B.K. Ray, M.Y. Wong, Orthogonal defect classification – a concept for in-process measurement, IEEE Transactions on Software Engineering 18 (11) (1992) 943–956.
[27] J. Coughlan, D.R. Macredie, Effective communication in requirements elicitation: a comparison of methodologies, Requirements Engineering Journal 7 (2) (2002) 47–60.
[28] W.J. Cristopher, The cost of errors in software development: evidence from industry, The Journal of System and Software 62 (1) (2002) 1–9.
[29] C. Debou, A.K. Combelles, Linking software process improvement to business strategies: experiences from industry, Journal of Software Process: Improvement and Practice 5 (1) (2000) 55–64.
[30] T. Dyba, Experiences of Undertaking Systematic Reviews, SINTEF ICT, Queensland, 2005.
[31] A. Endres, An analysis of errors and their causes in system programs, IEEE Transactions on Software Engineering 1 (2) (1975) 140–149.
[32] A. Endres, D. Rombach, A Handbook of Software and Systems Engineering, first ed., Pearson Addison Wesley, Harlow, England, 2003.
[33] M.R. Endsley, Situation awareness and human error: designing to support human performance, in: Proceedings of the High Consequence Systems Surety Conference, SA Technologies, Albuquerque, NM, 1999, pp. 2–9.
[34] N.E. Fenton, M. Neil, A critique of software defect prediction models, IEEE Transactions on Software Engineering 25 (5) (1999) 675–689.
[35] P.M. Fitts, R.E. Jones, Analysis of factors contributing to 460 'pilot error' experiences in operating aircrafts control, in: Proceedings of Selected Papers on Human Factors in the Design and Use of Control Systems, Dover Publications Inc., New York, 1961, pp. 332–358.

[36] W.A. Florac, Software Quality Measurement: A Framework for Counting Problems and Defects, CMU/SEI-92-TR-22, Carnegie Mellon Software Engineering Institute, Pittsburgh, PA, 1992, <http://citeseer.ist.psu.edu/florac92software.html>.
[37] B. Freimut, C. Denger, M. Ketterer, An industrial case study of implementing and validating defect classification for process improvement and quality management, in: Proceedings of the 11th IEEE International Software Metrics Symposium, IEEE Press, 2005.
[38] P. Fusaro, F. Lanubile, G. Visaggio, A replicated experiment to assess requirements inspection techniques, Journal of Empirical Software Engineering 2 (1) (1997) 39–57.
[39] D.A. Gaitros, Common errors in large software development projects, The Journal of Defense Software Engineering 12 (6) (2004) 21–25.
[40] J. Galliers, S. Minocha, A. Sutcliffe, A causal model of human error for safety critical user interface design, ACM Transactions on Computer–Human Interaction 5 (3) (1998) 756–769.
[41] R.B. Grady, Software failure analysis for high-return process improvement, Hewlett–Packard Journal 47 (4) (1996) 15–24.
[42] T. Hall, S. Beecham, A. Rainer, Requirement problems in twelve software companies: an empirical analysis, IEE Proceedings Software 149 (5) (2002) 153–160.
[43] J.H. Hayes, Building a requirement fault taxonomy: experiences from a NASA verification and validation research project, in: Proceedings of the 14th International Symposium on Software Reliability Engineering, IEEE Computer Society, 2003, pp. 49–59.
[44] J.H. Hayes, E.A. Holbrook, I. Raphael, D.M. Pruett, Fault-based analysis: how history can help improve performance and dependability requirements for high assurance systems, in: Proceedings of Fifth International Workshop on Requirements for High Assurance Systems (RHAS'05), IEEE Computer Society, Chicago, 2005.
[45] A. Issac, S.T. Shorrock, R. Kennedy, B. Kirwan, H. Andersen, T. Bove, The Human Error in ATM Technique (HERA-JANUS), HRS/HSP-002-REP-03, European Air Traffic Management, 2002.
[46] J. Jacobs, J.V. Moll, P. Krause, R. Kusters, J. Trienekens, A. Brombacher, Exploring defect causes in products developed by virtual teams, Journal of Information and Software Technology 47 (6) (2005) 399–410.
[47] S.H. Kan, V.R. Basili, L.N. Shapiro, Software quality: an overview from the perspective of total quality management, IBM Systems Journal 33 (1) (1994) 4–19.
[48] B. Kitchenham, Procedures for Performing Systematic Reviews, Technical Report TR/SE-0401, Department of Computer Science, Keele University and National ICT, Australia, Ltd., 2004, <http://www.elsevier.com/framework_products/promis_misc/inf-systrev.pdf>.
[49] B. Kitchenham, E. Mendes, G.H. Travassos, A systematic review of cross- vs. within-company cost estimation studies, in: Proceedings of 10th International Conference on Evaluation and Assessment in Software Engineering (EASE'06), Keele University, 2006.
[50] A.J. Ko, B.A. Myers, Development and evaluation of a model of programming errors, in: Proceedings of IEEE Symposium on Human Centric Computing Languages and Environments, IEEE Computer Society, 2003, pp. 7–14.
[51] A.J. Ko, B.A. Myers, A framework and methodology for studying the causes of software errors in programming systems, Journal of Visual Languages and Computing 16 (2) (2005) 41–84.
[52] J. Kramer, A.L. Wolf, in: Succeedings of the Eighth International Workshop on Software Specification and Design, ACM SIGSOFT Software Engineering Notes, vol. 21, no. 5, 1996, pp. 21–35.
[53] F. Lanubile, F. Shull, V.R. Basili, Experimenting with error abstraction in requirements documents, in: Proceedings of Fifth International Software Metrics Symposium, METRICS'98, IEEE Computer Society, Bethesda, MD, 1998, pp. 114–121.
[54] C.P. Lawrence, I. Kosuke, Design error classification and knowledge management, Journal of Knowledge Management Practice 10 (9) (2004) 72–81.
[55] M. Leszak, D.E. Perry, D. Stoll, A case study in root cause defect analysis, in: Proceedings of the 22nd International Conference on Software Engineering, ACM Press, Limerick, Ireland, 2000, pp. 428–437.
[56] R.R. Lutz, Analyzing software requirements errors in safety-critical, embedded systems, in: Proceedings of the IEEE International Symposium on Requirements Engineering, IEEE Computer Society Press, San Diego, CA, USA, 1993, pp. 126–133.
[57] R.R. Lutz, Targeting safety-related errors during software requirements analysis, The Journal of Systems and Software 34 (3) (1996) 223–230.
[58] J. Martin, W.T. Tsai, N-fold inspection: a requirement analysis technique, Communications of the ACM 33 (2) (1990) 225–232.
[59] C. Masuck, Incorporating a fault categorization and analysis process in the software build cycle, Journal of Computing Sciences in Colleges 20 (5) (2005) 239–248.
[60] R.G. Mays, C.L. Jones, G.J. Holloway, D.P. Studinski, Experiences with defect prevention, IBM Systems Journal 29 (1) (1990) 4–32.
[61] T. Nakashima, M. Oyama, H. Hisada, N. Ishii, Analysis of software bug causes and its prevention, Journal of Information and Software Technology 41 (15) (1999) 1059–1068.
[62] D. Norman, The Psychology of Every Day Things, Basic Books, New York, 1988.
[63] D.A. Norman, Steps towards a cognitive engineering: design rules based on analyses of human error, Communications of the ACM 26 (4) (1981) 254–258.
[64] D.A. Norman, Design rules based on analyses of human error, Communications of the ACM 26 (4) (1983) 254–258.
[65] K.M. Oliveira, F. Zlot, A.R. Rocha, G.H. Travassos, C. Galotta, C.S. Menezes, Domain-oriented software development environment, The Journal of Systems and Software 72 (2) (2004) 145–161.
[66] F. Paterno, C. Santoro, Preventing user errors by systematic analysis of deviations from the system task model, International Journal of Human–Computer Studies 56 (2) (2002) 225–245.
[67] S.L. Pfleeger, J.M. Atlee, Software Engineering Theory and Practice, third ed., Prentice Hall, Upper Saddle River, NJ, 2006.
[68] A.A. Porter, L.G. Votta, V.R. Basili, Comparing detection methods for software requirements inspections: a replicated experiment, IEEE Transactions on Software Engineering 21 (6) (1995) 563–575.
[69] J. Reason, Human Error, Cambridge University Press, Cambridge, USA, 1990.
[70] S. Sakthivel, A survey of requirement verification techniques, Journal of Information Technology 6 (2) (1991) 68–79.
[71] K. Sasao, J. Reason, Team errors: definition and taxonomy, Journal of Reliability Engineering and System Safety 65 (1) (1999) 1–9.
[72] G.M. Schneider, J. Martin, W.T. Tsai, An experimental study of fault detection in user requirements documents, ACM Transactions on Software Engineering and Methodology 1 (2) (1992) 188–204.
[73] L.W. Senders, N.P. Moray, Human Error: Cause, Prediction, and Reduction, Lawrence Erlbaum, Hillslade, NJ, 1991.
[74] S.T. Shorrock, B. Kirwan, Development and application of a human error identification tool for air traffic control, Journal of Applied Ergonomics 33 (4) (2002) 319–336.
[75] F. Shull, I. Rus, V. Basili, How perspective based reading can improve requirement inspection, IEEE Computer 33 (7) (2000) 73–79.
[76] J. Smith, The 40 root causes of troubled IT projects, Journal of IEEE Computer and Control Engineering 13 (3) (2002) 109–112.
[77] I. Sommerville, Software Engineering, eighth ed., Addison Wesley, Harlow, England, 2007.
[78] N.A. Stanton, S.V. Stevenage, Learning to predict human error: issues of acceptability, reliability and validity, Journal of Ergonomics 41 (11) (1998) 1737–1756.
[79] M.A. Stutzke, C.S. Smidts, A stochastic model of fault introduction and removal during software development, IEEE Transactions on Reliability 50 (20) (2001) 184–193.
[80] A. Sutcliffe, G. Rugg, A taxonomy of error types for failure analysis and risk assessment, International Journal of Human–Computer Interaction 10 (4) (1998) 381–405.
[81] A. Sutcliffe, A. Economou, P. Markis, Tracing requirements errors to problems in the requirements engineering process, Requirements Engineering Journal 4 (3) (1999) 134–151.
[82] A. Sutcliffe, B. Gault, N. Maiden, ISRE: immersive scenario-based requirements engineering with virtual prototypes, Requirements Engineering Journal 10 (1) (2004) 95–111.
[83] A. Swain, H. Guttman, Handbook of Human Reliability Analysis with Emphasis on Nuclear Power Plant Applications, Nuclear Regulatory Commission, Washington, DC, 1983.
[84] T. Thelin, P. Runeson, Robust estimation of fault content with capture–recapture and detection profile estimators, The Journal of Systems and Software 52 (2) (2000) 139–148.
[85] T. Thelin, P. Runeson, C. Wohlin, T. Olsson, C. Andersson, Team based fault content estimation in the software inspection process, in: Proceedings of the 26th International Conference on Software Engineering (ICSE'04), IEEE Computer Society, 2004, pp. 263–272.
[86] C. Trevor, S. Jim, C. Judith, K. Brain, Human Error in Software Generation Process, University of Technology, Loughborough, England, 1994, <http://www.branchlines.org.uk/Research/Tread1.pdf>.
[87] S. Viller, J. Bowers, T. Rodden, Human Factors in Requirement Engineering: A Survey of Human Sciences Literature Relevant to the Improvement of Dependable Systems Development Processes, Cooperative Systems Engineering Group Technical Report, CSEG/8/1997, Computing Department, Lancaster University, Lancaster, 1997, <http://citeseer.ist.psu.edu/viller97human.html>.
[88] G. Walia, J. Carver, T. Philip, Requirement error abstraction and classification: a control group replicated study, in: 18th IEEE International Symposium on Software Reliability Engineering, Trollhättan, Sweden, 2007.
[89] G.S. Walia, Empirical Validation of Requirement Error Abstraction and Classification: A Multidisciplinary Approach, M.S. Thesis, Computer Science and Engineering, Mississippi, Starkville, 2006.
[90] G.S. Walia, J. Carver, T. Philip, Requirement error abstraction and classification: an empirical study, in: Proceedings of IEEE Symposium on Empirical Software Engineering, ACM Press, Brazil, 2006, pp. 336–345.
[91] G.S. Walia, J. Carver, Development of a Requirement Error Taxonomy as a Quality Improvement Approach: A Systematic Literature Review MSU-070404, Department of Computer Science and Engineering, Mississippi State University, 2007, <http://www.cse.msstate.edu/PUBLICATIONS/TECHNICAL_REPORTS/MSU-070404.pdf>.
[92] X. Zhang, H. Pham, An analysis of factors affecting software reliability, The Journal of Systems and Software 50 (1) (2000) 43–56.