

# A SYSTEMATIC LITERATURE REVIEW TO IDENTIFY AND CLASSIFY SOFTWARE REQUIREMENT ERRORS

Technical Report MSU-071207  
Gursimran Singh Walia and Jeffrey C. Carver

## **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 understand whether using information about the source of a fault (i.e. and error) can be helpful. Once the usefulness of errors is established, then it is important to identify and classify errors. The review identified 149 papers from software engineering, psychology and human cognition that provided 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.*

*Keywords – Systematic Literature Review, Human Errors, Software Quality*

## **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, 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 [18, 27].

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 the paper. These definitions are quoted from Lanubile, et al. [51], and are consistent with software engineering textbooks [31, 65, 75] 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 life cycle). 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 [18, 27, 84]. 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 [47, 77, 90]. To be effective, a systematic review must be driven by an overall goal. In this review, the high level goal is to:

*Identify and document the types of requirement errors in a manner that will improve overall quality.*

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 to highlight the areas of need. Then Section 2.2 introduces the concept of *error abstraction* and describe the literature that was examined during the 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, 8]. 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 [35]. 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 [53]. Building on Root Cause Analysis, the Orthogonal Defect Classification (ODC) was developed to provide in-process feedback to developers and help them learn from their mistakes. The ODC also uses an understanding of the faults to identify cause-effect relationships [15, 25]. In addition, fault classification taxonomies have been developed and evaluated to aid in quality improvement throughout the development process [36, 42, 57].

Empirical evidence suggests that “quantifying, classifying and locating individual faults is a subjective and intricate notion, especially during the requirement phase” [37, 51]. 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) [52]. These fault classification taxonomies have been the basis for most of the software inspection techniques (e.g., checklist-based, fault-based, perspective-based) used by developers [9, 21, 24, 37, 56, 73]. However, because these techniques do not focus on errors, inspectors still leave some faults undetected. To compensate for these undetected faults, various supporting mechanisms have been added to the quality improvement process to try to improve their overall effectiveness, e.g., re-inspections, and defect estimation techniques [4, 10, 33, 82, 83].

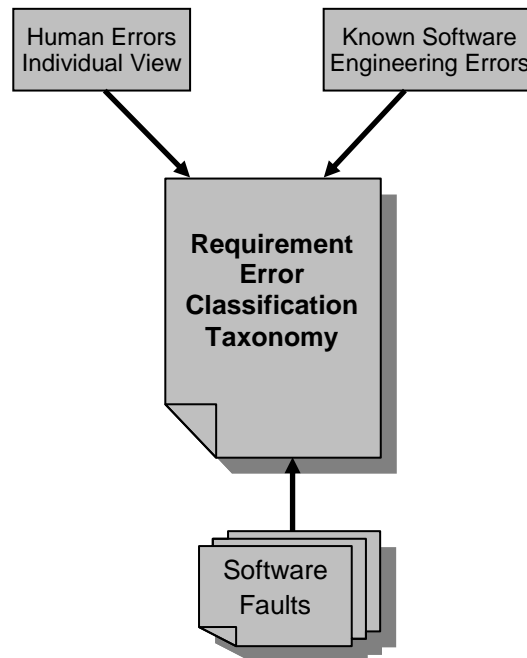
The above research efforts have made significant contributions to fault reduction; however, requirements faults still slip to later development phases. This fault slippage motivates a need for additional information to support the software quality process. The use of error information is a promising approach for providing such insight. Therefore, 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.

## 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 [51]. 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 [53, 84]. Studies of these human reasons can be found in research from the fields of human cognition, and psychology.

Therefore as Figure 1 shows, literature from software engineering, human cognition, and psychology are needed to provide a more comprehensive list of errors that may occur.



**Figure 1 – Types of Literature Searched**

### 3. RESEARCH METHOD

The systematic review reported in this paper was done following the guidelines of Mian, et. al., Kitchenham, et. al., and Dyba [41, 77, 99]. The steps involved in the review included: formulating a review protocol, conducting the review, analyzing the results, reporting the results, and discussing the findings. Conducting the review included: identifying primary studies, evaluating those studies using inclusion and exclusion criteria, quality assessment, extracting data from the selected studies, and synthesizing that data into a concrete result.

Prior to the review, a protocol was developed that specified the questions to be addressed, the databases to be searched and the methods to be used to identify, collate, and assess the evidence. To reduce researcher bias, the protocol, described in the remainder of this section, was developed by one of the authors, reviewed by the other author and then finalized through discussion, review, and iteration among the authors and their research group.

#### 3.1 Research Questions

The main goal of this systematic review was to identify and classify different types of requirement errors and to describe their likely impact on software quality in terms of faults. 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?*

This high-level question was then decomposed into the more specific research questions and sub-questions shown in Table 1.

Table 1- Research Questions and Motivations

<b>Research Question</b>	<b>Motivation</b>
1. Is there any evidence that using error information can improve software quality? <i>1.1. Are there any processes or methods reported in literature that use error information to improve software quality?</i> <i>1.2. Do any of these processes address the limitations and gaps identified in Section 2 of this paper?</i>	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? <i>2.1. What types of errors can occur during the requirement stage?</i> <i>2.2. What errors can occur in other phases of the software lifecycle process that are related to errors that can occur during the requirements phase?</i>	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 <i>3.1. What information can be found about human errors and their classification?</i> <i>3.2. Which of the human errors identified in Question 3.1 can have corresponding errors in software requirements?</i>	Investigate the contribution of human errors from the fields of human cognition and psychology
4. <i>How can the errors be grouped into a requirement error classification taxonomy?</i>	Organize the information into a taxonomy that can be easily used by developers

### 3.2 Source Selection and Search

Prior to conducting the search, the correct set of databases must be selected to optimize the likelihood of finding the most complete and relevant sources. In this review, the following criteria were used to select the source databases:

- The databases were chosen to include journals and conference proceedings that cover: software quality, software engineering, empirical studies, human cognition, and psychology;
- If available, 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;
- The databases of any relevant journals, proceedings or other literature that were not included in one of the above databases were searched separately;
- The list of databases was then reviewed by experts from software engineering and cognitive psychology who augmented the list with additional databases. Prior to conducting the actual search, the researchers involved agreed on the final list of databases;
- The list of databases was reduced where possible to minimize the redundancy of journals and proceedings across databases.

**Table 2 – Source List**

<b>Databases</b>	<b>Other Journals or Conference Proceedings</b> (not covered by databases)	<b>Additional Sources</b>
IEEEExplore	Empirical Software Engineering – An International Journal, Requirement Engineering Journal	Reference lists from primary studies and other review articles
INSPEC		Books
ACM Digital Library		SEI technical report website
SCIRUS (Elsevier)		
Google Scholar		
PsychINFO (EBSCO)		
Science Citation Index		

Based on the criteria for selecting database sources (defined earlier in this section), an initial list of sources was created. A software engineering expert and a cognitive psychology expert examined this list. As a result, two databases and one journal were added to the list. The final source list is shown in Table 2.

To search these databases, a set of search strings was created for each research question based on keywords extracted from the research questions and augmented with synonyms. Table 3 shows the search strings used for each research question. In addition, some key journals and conferences were searched directly to ensure that all relevant literature was found.

In developing the keyword strings to use when searching the source databases, 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 3 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); *review question* – maps to the primary research questions; *purpose* – why the search string was included.

Table 3 - Search Strings

String #	High Level Search String	Detailed Search String	Review Question	Purpose
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))	Q1	To determine if there are any quality improvement approaches that make use of error information
2	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))	Q2	To locate types of requirement errors
3	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))	Q2	To find any existing error taxonomies, and to abstract errors from existing fault taxonomies that are relevant to requirements
4	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))	Q1	To determine if there are any inspection methods that focus on error information (specifically in requirement specifications)
5	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)).	Q3	To review the literature about human errors and their classifications to identify errors that can occur at the requirement stage
6	Contribution from Diverse Research fields to Software Development	((contribution OR significance OR benefit OR supplement OR assistance) AND (human cognition OR psychology) AND (software development)).	Q3	To review additional literature about human errors
7	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))	Q1	To supplement search String 1

Performing the searches from Table 3 on the databases in Table 2 resulted in an extensive list of potential papers that could be included in the review. To ensure that only the most relevant papers were included, a set of detailed inclusion and exclusion criteria was defined (Table 4).

**Table 4- Inclusion and Exclusion Criteria**

Inclusion Criteria	Exclusion Criteria
<ul style="list-style-type: none"> <li>• Papers that talk about errors, mistakes or problems in the software development process and requirements in particular</li> <li>• Papers about quality improvement approaches including inspection techniques</li> <li>• Papers about error, fault, or defect classifications, especially for requirements</li> <li>• Papers from diverse research fields of human cognition, psychology that talk about human errors and their classifications</li> <li>• Empirical studies (qualitative or quantitative)</li> <li>• Other papers that directly address the research questions</li> </ul>	<ul style="list-style-type: none"> <li>• Papers that are based only on expert opinion</li> <li>• Short papers, introductions to special issues, tutorials, and mini-tracks</li> <li>• Studies not related to any of the research questions</li> <li>• Preliminary conference versions of included journal papers</li> <li>• Studies presented in language other than English</li> <li>• Studies whose findings are unclear and ambiguous</li> </ul>

Using these criteria, the results of the database searches were examined to arrive at the final list of papers. 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 additional papers not related to the research focus
3. Read the remaining papers and eliminate any that are not related to the research questions.

This processes resulted in a list of papers to include in the systematic review. Appendix A provides a summary of each of the included papers.

After using the inclusion and exclusion criterion to select relevant papers, a quality assessment was performed on those papers. This quality assessment was another check on the quality of the set of papers that resulted from the initial search. Quality assessment was done in accordance to CRD guidelines as cited by Kitchenham et al; i.e., each study was assessed for bias, internal validity and external validity of the results [77]. First, the study described in each paper was classified as either an experiment or an observational study based on the guidelines. For each experiment and observational study, a set of questions (quality criterion) was used to evaluate the quality of the study (see Table 5). The questions focus on study design, bias, validity, and generalizability of the results.

**Table 5 - Paper Quality Assessment**

Questions	Experimental Studies	Observational Studies
1	Does the evidence support the findings?	Do the observations support the conclusions or arguments?
2	Does study identify and try to minimize biases?	Does study uses methods to minimize biases?
3	Does the study mention internal and external validity threats?	Were different intervention described accurately?
4	Was the analysis appropriate?	Are comparisons clear and valid?
5	Can the study results be generalized?	Can the study results be generalized?
6	Can this study be replicated?	Can this study be replicated?

Our search returned over 25,000 total papers, which were narrowed down to 7838 papers based on their titles, and 482 papers based on their abstracts and keywords. Then, these 482 papers were read to select a final list of 149 papers using the inclusion and exclusion criterion (Table 4). Of these 149 papers, 108 are published in thirteen leading journals and six conferences in software engineering, and 41 are 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.

Using the quality assessment, 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 [15, 23, 25, 27, 51, 53, 67] (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 these journals seemed like they should have a larger number of relevant papers, we wanted to ensure there was not a problem with the database being used, so we 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, therefore the numbers from some conferences are lower than expected in Table 6 (e.g. some preliminary work by Sutcliff, 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* [78])

**Table 6 - Paper Distribution**

<b>Source</b>	<b>Count</b>	<b>%</b>
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 & 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%
<b>Total</b>	<b>149</b>	<b>100%</b>

### 3.3 Data Extraction

To ensure consistent and accurate information extraction from the papers, data extraction forms were created and reviewed during the development of the study protocol. The data

extraction forms were used to collect the information needed to address the review questions. Due to the different foci of the review questions multiple data extraction forms were created. One form was created that contained information common to all research questions and extracted from all papers (Table 11). Additional forms were created to extract information specific to each search string only from papers that were related to that search string (Table 12). One of the authors reviewed all of the papers and extracted the information using the data extraction forms. As done in other systematic review, the other author reviewed a sampling of the papers and also extracted information using the data extraction forms [71]. The results of the two authors were compared to ensure there were no significant differences. The two authors consistently extracted information from the small sample, and therefore determined that the second author did not need to also review all of the papers. Table 11 and Table 12 are provided in Appendix B.

### 3.4 Data Synthesis

The data synthesis focused on collating and summarizing the data extracted from the papers to address the research questions from Section 3.1 that cover: quality improvement approaches, requirement errors, multidisciplinary review, inspection methods and error-fault-defect taxonomies. A summary of the results for each research focus is provided in its own table in Appendix C. The following subsections describe the characteristics of each table including the search strings used, the research questions addressed, and the relevance to the research goals.

#### 3.4.1 Software Quality Improvement Approaches (Table 13)

This table describes techniques, methods, or processes that can improve software quality. Software quality improvement is very broad, so this search focused on those quality improvement approaches that used error information. The goal of this search was to locate quality improvement approaches that use error information and record any information about specific software errors. This search used search strings 1 and 7 (as shown in Table 3 ) to address research question 1. Table 13 describes the quality improvement approaches located that use error information. Each quality improvement approach is characterized by: authors, name of the approach, main characteristics, error focus, limitations, relevance, and the list of related references.

#### 3.4.2 Software Requirement Errors (Table 14)

This table describes errors, problems, or mistakes that can occur at requirements stage of the software process. This search used search string# 2 and 3 (as shown in Table 3) and focused on research question 2. The search was very effective in gathering a lot of information about errors and mistakes that can occur during development of the requirements specification, and other root causes of faults traceable to requirements stage. Table 14 summarizes the results characterizing each source by: authors, a description of the study, the high-level error types, and the kinds of requirement errors emphasized, the study's relevance, and the list of related references. This search also located studies describing errors that can occur at other stages of software process (using search string 3). These studies (especially those about design and coding errors) are also included in Table 14 because they were used to augment requirement error information.

### 3.4.3 Review of Literature from Other Domains (Table 15)

An exhaustive review of human errors and error classifications in the fields of human cognition and psychology was conducted to understand human errors & fallibilities, to gather information about different human errors and to determine whether those errors can occur during requirement stage. This search used search strings 5 and 6 (as shown in Table 3) and focused on research question 3. Table 15 presents the human error results characterized by: author, name of the classification, description of the error classification, and the list of related references. The errors in each of these classifications were analyzed to determine not only which could occur at the requirement stage but also the types of faults that could result. This search was also quite effective in terms of the number of relevant studies found. Furthermore, human cognition research has focused on understanding human errors. Surveys of human error classifications have been published. These surveys on human error classifications provided references for relevant classifications. The details of these classifications are provided in Table 15.

### 3.4.4 Software Inspection Techniques (Table 16)

The goal of this search was to analyze different software inspection methods to determine if any used error information. This search used search string 4 (as shown in Table 3) and focused on research question 1. Table 16 gives the summary of the results that provided a lot of information about different inspection methods, including some existing surveys of inspection literature. The inspections techniques are characterized by: author, name of the technique, application context, a description, its error focus (e.g. does the technique focus on errors?), and the list of related references.

### 3.4.5 Error-Fault-Defect Taxonomies (Table 17)

The goal of this search was to gather information about requirements defects to search for patterns and allow the causes of defects to be abstracted to the underlying errors. This search used search string 3 (as shown in Table 3). The results, shown in Table 17 summarize the fault classifications that were analyzed. The information of errors abstracted from these faults was also used to augment the requirements error information. Each classification is characterized in by: author, classification name, application context, description and the list of related references.

## 4. Reporting the Review

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

A review of the literature did indicate that using error information during software inspections can improve quality. Knowledge of the source of faults is useful for software process improvement in real-time, defect prevention, quality management, learning, and packaging experiences. The approaches that emphasize the use of error information typically do not provide any 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 [51]. While each approach has some positive aspects, they also have limitations as discussed in the answers to Questions 1.1 and 1.2. A total of thirteen papers were used to address this question and its related sub-questions.

**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. Unfortunately, none of these methods provide detailed guidance to a developer. The strengths and limitations of each method are summarized below.

- The *defect causal analysis* approach is a team-based quality improvement technique used to analyze previous faults to determine their cause (the error) and prevent future faults [23, 40]. The main limitations of this method are its extensive documentation requirement, its need for experienced developers, and its reliance on a sampling of the faults, potentially overlooking some errors. Also, the error categorization is too generic and appears to be incomplete.
- 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 [45]. The main limitations to this approach are its reliance on expert knowledge rather than actual fault data and the extensive meeting requirement for the experts to analyze probable causes.
- The *defect prevention process* uses causal analysis to determine the source of a fault and to suggest preventive actions. However, past experience with causal analysis reveals that it is cost intensive, people intensive and useful for analyzing only a small sample of faults [58]. Therefore, it is a partial feedback mechanism that does not consider all faults [15]. While, this approach identifies a few error categories, they also appear to be too general and incomplete.
- The *software bug analysis* process identifies the source of bugs (faults). It also contains countermeasures (with implementation guidance) for preventing bugs [59]. Rather than focusing on requirements faults, this analysis focuses on faults from late in the life cycle, and it does not represent all errors.
- The *root cause analysis* method helps developers determine the actual causes of a fault (the error) [53]. The main limitations to this approach are that it requires a lot of time, training, and expertise, and a large number of actions. Also, because each fault is analyzed individually, this approach is not very effective for analyzing large collections of faults. An efficient method is needed for analyzing the cause(s) of group of related faults.
- 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 [51]. The main limitation to this approach is the lack of an error taxonomy to guide the inspector. Rather, it relies heavily on the creativity of the inspector in identifying errors.
- 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 [23, 57]. The main limitation to this process is that it analyzes faults late in the lifecycle (e.g., design and code). Another limitation is that the written description that is required for analyzing a fault cannot be processed by attribute focusing.
- 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 [7]. This approach does describe some errors that occur in the human thought process, but its main limitations include process feedback problems (i.e., it can only detect process

inadequacy not reveal the actual cause), over-reliance on historical data, and difficulty detecting faults that are unique to the current project.

- *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 [46]. This approach is based on the *defect prevention process* (described earlier) and it has the same limitations.

Even though each approach individually has some benefits and limitations, many of them do identify some important types of errors. This information about requirement errors and other process errors served as an input to the analysis for *Question 4*.

Table 7 – 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

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

The quality improvement approaches discussed in Sections 1 and 2 have limitations in their ability to ensure software quality as listed in Table 7. The processes described in *Question 1.1* do not fully address these limitations. While some of them do focus on errors rather than just faults (addressing one of the limitations), they each suffer from other limitations, preventing them from being a complete solution. The processes use different types of causal analysis methods to develop error categories. Because the resulting error categories are incomplete, they leave gaps in the errors and related faults that the developers are guided towards. The *defect prevention process* addresses the limitation that developers do not learn from their mistakes; unfortunately, it uses only a partial feedback mechanism that does not identify all of the faults. The inability of the existing methods to overcome these limitations motivates the need for the development of additional approaches.

**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 already described by other researchers. This research question is addressed in detail by the research questions 2.1 and 2.2. A total of fifty five papers were analyzed to identify the types of requirement errors in software engineering literature. Out of them, thirty one were used to address question 2.1 and twenty four were used to address question 2.2.

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

The review uncovered a number of candidates for requirement errors. Table 8 lists the sources of potential requirement errors along with the relevant references.

Table 8 – Sources used to Identify Requirement Errors in the Literature

Sources of Errors	References
Root causes, cause categories, bug causes, defect-fault causes	[15, 23, 40, 45, 58, 59]
Requirement engineering problem classification	[12, 20, 41, 75, 79, 80]
Empirical studies on root causes of troubled projects or errors	[6, 74]
Causes of requirement traceability and requirement inconsistency	[6, 28, 32, 63, 69]
Domain knowledge problems	[63]
Management problems	[28]
Situation Awareness / Decision making errors	[32]
Team errors	[69]
Influencing factors in req. stage and software development	[11, 26, 77, 90]
Other	[38, 54, 55, 77, 85, 90]

**Question 2.2:** *What errors can occur in other phases of the software lifecycle process 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. Some of those 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 [16, 40].
- *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 chose 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 [61, 62].
- *System programs:* Technological, organizational, historical, individual or other causes [30].
- *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 [49, 50].

In addition to identifying requirement errors directly from the literature, this review also identified a list of faults that could be traced back to their underlying error [3, 5, 13, 14, 25, 36, 43, 66, 68, 70]. These errors were added to the list of errors that served as input to *Question 4*. Examples of the errors include:

- Misunderstanding or mistakes in resolving conflicts (e.g., there are unresolved requirements or incorrect requirements that were agreed on by all parties);
- Lack of participation of all stakeholders in the requirements process;
- Mistakes or misunderstandings in mapping inputs to outputs, input space to processes, or processes to output; and
- Unresolved issues about complex system interfaces or unanticipated dependencies.

**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 thirty two papers addressed the use of human errors in software requirements phase.

**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) [67].
- *Rasmussen's skill, rule and knowledge based human error taxonomy:* Skill based slips and lapses are caused 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 [67].
- *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 [67].
- *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, attention), and errors based on biases [71].
- *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), or timing errors (something is done too early or too late) [81].
- *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 [34].

- *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 [22].
- *Galliers, Minocha, and Sutcliffe's taxonomy of influencing factors for occurrence of errors*: environmental conditions, management & organizational factors, task/domain factors, and user/personnel qualities including the slip and mistake types of errors described earlier [39].
- *Sutcliffe and Rugg's error categories*: operational description, cognitive causal categories, social and organizational causes, and design errors [78].
- *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 [60-62, 64].
- *Human error identification (HEI) tool*: Describes the SHERPA tool that classifies errors as action, checking, retrieval, communication or selection [72, 76].
- *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 & vigilance, working memory, long-term memory, judgment, planning & decision-making, and response execution [19, 44].

Table 9 - Requirement Errors Drawn from Human Errors

<b>Error</b>	<b>Description</b>
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

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

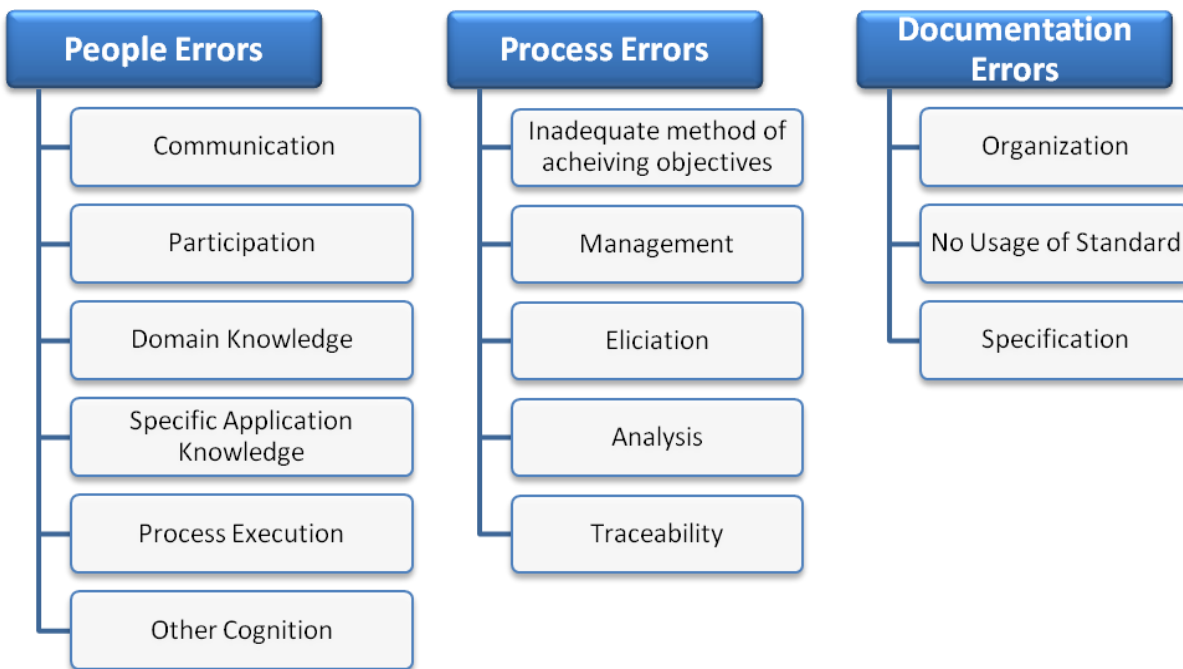
Those errors that were relevant to requirements were included in the initial list of errors that served as input to *Question 4* to make it more comprehensive. Examples of the translation of these errors into requirements errors are found in Table 9.

**Question 4:** How can the errors be grouped into a requirement error classification taxonomy?

Some of the errors were identified in more than one of bodies 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 details of the Requirement Error Taxonomy are discussed in Section 4.

## 5. REQUIREMENT ERROR TAXONOMY

We classified the errors into a taxonomy to provide guidance for future error detection research with the intent of addressing limitations in the existing quality improvement approaches. The errors identified from software engineering and psychology fields were collected, analyzed for similarities, and grouped together into the taxonomy. Errors from both software engineering and psychology that had similar characteristics (symptoms) were grouped together to aid in a more thorough ability to check for related errors, when one error is found.



**Figure 2 – Requirement Error Taxonomy**

The taxonomy was developed based on the concept of a 2-level view (i.e. the high level classes of People, Process, and Documentation contain more detailed error classes) to aid its understanding and usability for developers.

The taxonomy organizes the errors into three high-level types: people errors, process errors, and documentation errors. Each type is then further refined into more detailed classes as shown in Figure 2. Each of these 14 error classes was created by grouping similar errors together. An important constraint while grouping the requirement errors was to keep the groups as orthogonal as possible. In addition, the taxonomy identifies faults that are likely to be caused by each error type [87, 88].

To explain each error class we first describe the error and then give an example of how the error might occur in one of the following applications. Due to space limitations, only a brief overview of each system is provided here. The errors illustrate problems that may occur in such a system.

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

## 5.1 People Errors

These errors concern individual human fallibilities that are the result of the failures of the people involved in the project development. These errors arise from various kinds of mistakes, misunderstandings or forgetfulness (e.g., lack of communication, misunderstanding of customer needs, and misunderstanding of the specific application). We identified six specific classes of People Errors.

### A. *Communication Errors*

- Lack of communication among stakeholders including communication with customers and users, communication within the team and communication between teams;
- Unclear lines of communication and authority structure, leading to a lack of consensus on technical standards and approaches;
- Lack of communication of changes made to a document;
- **Example:**
  - *Error*: The customers do not communicate all the requirements (e.g., the requirement that an ATM system should dispense different types of currency such as US dollars and Euros was not clearly specified by the customer).

- *Fault*: Omitted functionality (i.e., The requirements for the ATM only specifies that it handle dollars. The capability to dispense other currencies is missing).

#### B. *Participation Errors*

- Lack of participation of all stakeholders (including all user groups) during development;
- Lack of motivation or rivalry among the parties involved in the project development;
- Lack of a mechanism to mediate conflicts among stakeholders;
- **Example:**
  - *Error*: A specific user (e.g., the bank manager) was not involved in the requirements process; and therefore, his needs have not been included (e.g., the requirement that an ATM system should support multiple copies of an ATM card simultaneously).
  - *Fault*: Omitted or incorrect functionality (i.e., the requirement specifying the use of multiple cards was omitted from the requirements).

#### C. *Domain Knowledge Errors*

- Lack of experience or domain knowledge on the part of the requirements author;
- Misunderstandings due to complex nature of task domain;
- Lack of skills required for performing a particular task (e.g., the person tasked with creating the requirements elicitation plan does not possess necessary knowledge);
- Some properties of the problem space are not investigated, leading to incorrect dependencies between pieces, wrong assumptions, or incorrect system behavior;
- Mistaken assumptions regarding system states, preconditions, and post-conditions;
- **Example:**
  - *Error*: The requirements author is not knowledgeable about a particular domain in which he must specify requirements (e.g., for an ambulance dispatch system the requirements author does not understand medical incidents, but must specify this information in the requirements);
  - *Fault*: Incorrect functionality (i.e., the requirements specify the incorrect algorithm for assignment of ambulances).

#### D. *Understanding Specific Application Errors* - These errors are caused by misunderstandings about the specific applications (as opposed to the general domain):

- Mistakes regarding the expression of the end state, output, goals, or objectives to be achieved;
- Misunderstanding or mistakes in resolving conflicts within the domain (e.g., there are unresolved requirements or bad requirements and all the parties accept the problems);
- Misunderstandings about the relationships and dependencies between individual parts of the system and the real world;
- Mistakes or misunderstandings about timing constraints or timing relationships among commands in concurrent process execution;
- Misunderstandings of the data dependency constraints or conflicts regarding the restrictions on command order when 2 or more processes access the same input;
- Misunderstandings of event or command ordering or functional properties;
- Misunderstandings of the software interfaces with the rest of the system or the hardware interfaces;

- Mistakes or misunderstandings in mapping the inputs to outputs, input space to processes, and processes to output;
- **Example:**
  - *Error:* Misunderstandings regarding the ordering of the events (e.g., in AAD, for providing “ambulance service”; an action that should be a precondition like setting the ambulance status data to current is specified as occurring after the incident is completed);
  - *Fault:* Wrong precedence relationship (i.e., the requirements specify the incorrect order of the events for providing “Ambulance Service”).

#### E. *Process Execution Errors*

- Mistakes in applying the process, regardless of whether it is adequate for the task at hand;
- Execution/storage mistakes, steps to achieve necessary functionality are out of order and lapses on the part of people executing the method;
- **Example:**
  - *Error:* Mistakes in the ordering of a particular step(s) (e.g., in the ATM system, there is a misunderstanding about the timing of the following steps: 1) returning the ATM card to the user, and 2) retaining the ATM card.)
  - *Fault:* Incorrect functionality (Requirements will incorrectly specify when the card should be ejected and when it should be kept. During an unsuccessful transaction, the users’ card is incorrectly kept by the ATM and an error message is displayed indicating that customer should call the bank – the card should be returned. Similarly, if an incorrect password is entered more than three times in a row, an error message is displayed and card is ejected – the card should be returned);

#### F. *Other Human Cognition Errors*

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

### 5.2 Process Errors

These errors are due to the inadequacy of the processes chosen for the requirement-engineering phase. The various processes include: planning, management, elicitation, and other processes.

#### A. *Inadequate Methods for Achieving Goals and Objectives Errors*

- Missing or inadequate setting of goals and objectives;
- Wrong method chosen because some system-specific information was omitted or misunderstood;
- Selection of a method that was successful on a previous project for use in a completely unknown situation without proper investigation;
- All the important facts were understood, but the wrong method was still chosen;
- **Example:**
  - *Error:* Mistake (wrong information used) in selecting a method (e.g., for selecting a method of “issuing warnings” in AAD, the requirement engineer uses the values of

- status data for deciding when to issue warning; however, the information used should be the values of the incident that is open and not progressing).
- *Fault*: Incorrect or infeasible processes (i.e., “Issuing Warning” functionality cannot be achieved with specified information).

#### B. *Management Process Errors*

- Omissions or misunderstandings about the assignment of resources to different development tasks;
- Lack of leadership and necessary motivation;
- Omission or misunderstanding of all the alternatives and their impact.
- **Example**:
  - *Error*: Mistakes in assignment of resources (e.g., in AAD system, in case of “review incidents,” the input and output resources are not allocated for batching up incident reports and receiving call from incident sites requirements);
  - *Fault*: Incompleteness and ambiguity in requirements due to unavailability of the resource requirements.

#### C. *Requirements Elicitation Process Errors*

- Lack of questionnaires or interviews for eliciting requirements from customer;
- Slips or lapses or lack of awareness of all sources of the requirements;
- Inadequate procedure for collecting requirements from various sources relevant to the problem domain;
- **Example**:
  - *Error*: Inability to elicit all the requirements (e.g., in LAS system, the developers are not able to elicit the requirements about response time for emergency incidents or error handling requirements).
  - *Fault*: Performance requirements and other non-functional requirements are omitted.

#### D. *Analysis Process Errors*

- Mistakes or misunderstandings in analysis of technical, operational, and financial feasibility and risks of requirements. This mistake can lead to unmet user needs and objectives;
- Mistakes in developing system models and scenarios for analyzing requirements;
- Mistakes, lapses, or misunderstanding while partitioning the system into manageable pieces for analysis leading to omission of pieces or the redundancy among pieces;
- Misunderstandings or unresolved issues about complex system interfaces and unanticipated dependencies;
- Lack understanding and representing input and output space for all types of execution in different circumstances;
- Inability to predict or guarantee the exact behavior for all kind of inputs and for different states;
- Misunderstanding of desired behavior of the software;
- **Example**:

- *Error*: Not identifying the exact behavior required during the requirements analysis (e.g., in PGCS, the system response to the driver takes a ticket and not entering the parking garage is not understood);
- *Fault*: The requirements omit the proper system response for the situation leaving it undefined and possibly erroneous..

#### E. Traceability Process Errors

- Inadequate means of achieving traceability of requirements to predecessors and successors;
- Inadequate change management, including analysis of impacts and tracking changing requirements;
- **Example**:
  - *Error*: Mistakes while establishing requirements traceability (e.g., in PG system, the ability to increase the number of reserved parking spaces cannot be traced to any user requirements);
  - *Fault*: Extraneous requirement (a requirement regarding to allow for increase in the number of spaces is included. It does not support any user functionality and can not be traced back to any abstract predecessor requirement therefore it is not necessary and could lead to additional work).

### 5.3 Documentation Errors

These errors occur while specifying the requirements, as understood by the requirements engineer, in a natural language document. These errors assume that the requirement engineer correctly understood the desires of the customer; therefore they are different from the people or process errors, because in this case, the mistake is made during the specification process. These are errors from the point of view of the requirements author and are the cause of the faults found while reading the document.

#### A. Requirements Organization Errors

- Lapses or mistakes in listing or organizing requirements in the document;
- Ineffective selection of method for organizing the individual requirements;
- **Example**:
  - *Error*: In PGCS, the requirements are listed without any logical organization.
  - *Fault*: Missing requirement (Method for grouping requirements does not consider all categories and so some requirements, e.g., requirement about synchronization of entry/exit gates is missed out and not documented as it does not belong to any requirement category as per selected method).

#### B. No Usage of a Standard

- No usage of standard for documenting requirement specification;
- No usage of standard tool like a checklist to ensure all important items are documented;
- No usage of standard for notational descriptions (i.e., different notations or formatting used for same object in different sections is a part of misunderstanding);
- **Example**:

- *Error*: In documenting the requirements for the ATM system, the IEEE standard was not used. Therefore the scope of the system and performance requirements was omitted.
- *Fault*: Omitted and incomplete requirements (i.e., the scope of the system and performance requirements are missed out).

### C. Specification Errors

- Mistakes or lapses while organizing the requirements regardless of the adequacy of the organization method (e.g., the requirement intended for one set is mistakenly put into other set that do not bear any similarity);
- Omission of necessary attention checks at critical points can lead to deviation from intention;
- Using extraneous checks leads to repetition/omission of steps;
- Large time gap between solution formulation and its application;
- Mistakes in referencing to incorrect sections / requirements;
- Lack of awareness in description of the performance requirement specification and/or “implementation constraints”;
- **Example**:
  - *Error*: The requirement author used a specific method to organize the requirements, but he still makes mistakes while creating the specification (e.g., in AAD system, the requirements that the error recovery functionality should have clearly defined states is intended to be placed in section of the document, but instead it is placed in the wrong section);
  - *Fault*: Mistakes in organizing can lead to ambiguity in requirements (e.g. in the AAD system, the requirements about error handling become ambiguous).

## 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 these findings to software engineering research community.

### 6.1 Principal Findings

The goal of this review was to develop a classification of the types of errors that can occur in the requirement phase. Thus, a systematic literature review was conducted across different research fields to identify types of requirements errors. Using this information, a comprehensive initial Requirement Error Taxonomy was developed. The principal findings of the review (discussed in Section 4) are:

- A description of different software quality improvement methods that use error information, their limitations, drawbacks, and contributions to the requirements error taxonomy. This information is summarized in Table 13 and Table 16.
- A description of the various requirement errors that have been reported in the software engineering literature. This information is provided in Table 14. In addition, an analysis of groups of related faults described in Table 17 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 of software process. This information is provided in Table 15.
- Finally, a Requirement Error Taxonomy is presented in Section 4 that classifies all of the errors uncovered during the systematic review along with their impact on software quality in terms of the faults they are likely to cause.

## 6.2 Strengths and Weaknesses

### 6.2.1 Strengths

**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 and accurately 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 result of the researchers' own assessment of the quality of the evidence indicated that the search evidence was sufficiently rigorous and addressed all the research issues. A further indication of the completeness of the search was the fact that a set of relevant papers, which were known to the researchers prior to performing the systematic review, were independently found during the systematic review. Finally, the results are based on papers that include empirical evidence rather than merely expert opinions.

**Ability of the Requirement Error Taxonomy to overcome limitations:** Revisiting the limitations and drawbacks identified in Sections 1 and 2 and in Question 1.2, the Requirement Error Taxonomy provides comprehensive error information that can help developers to identify and classify problems at their origin thereby identifying the actual causes of problems (errors) in software development. The description of different types of mistakes provides some insight into the major problem areas to help developers to learn from their mistakes. Using the Requirement Error Taxonomy should enable developers to uncover a greater percentage of the errors and related faults present in a requirements document than is possible with the existing approaches surveyed (although further empirical evaluation of this concept is underway). Finally, the Requirement Error Taxonomy was developed to meet certain essential criteria: simplicity, understandability, applicability, intuitiveness, orthogonal, comprehensiveness, usefulness, and uniformity across products.

### 6.2.2 Weaknesses:

The main weakness of the output of this systematic review is the lack of empirical validation of the validity and usefulness of the Requirement Error Taxonomy. Rather, the discussion in this paper is based on the knowledge of the limitations of existing approaches and the potential that the Requirement Error Taxonomy has for overcoming those limitations. An initial study was performed with university students that provided some positive initial results [145], but further validation is needed.

### 6.3 Contribution to Research and Practice Communities:

The result of this research provides a new perspective into the investigation of software quality improvement. Research in this paper 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 is conducted to develop an error-based approach to overcome some of these limitations. The result of the research is the development of a Requirement Error Taxonomy that will help developers to identify software development problems at their origin efficiently. Initial investigation of the error abstraction approach using Requirement Error Taxonomy has also provided improvements to software quality [145]. Further investigation into the proposed error based approach will help research and practice community use this error information to identify the root cause of problem efficiently. In addition, understanding of the actual causes of the problems in software development will help developers learn from their mistakes and develop preventive measures to prevent them from occurring in the future. The Research presented here will help motivate the software community to use tools and techniques based on software errors.

### 6.4 Conclusion and Future Work

Based on the evidence gathered from the review and the data collected from an initial study, the proposed error abstraction approach and Requirement Error Taxonomy provide a lot of promise to the software developers. The evidence provided in this paper has motivated further investigation of the effectiveness of the proposed taxonomy. So, the future work consists of validating the Requirement Error Taxonomy through further empirical studies. These results will motivate interested researchers to further investigate this area to develop more effective tools and methods to assist software developers.

Also conducting similar systematic reviews to identify errors in the subsequent software lifecycle phases will help software organizations to use this approach throughout the software development life cycle. The continuing research in this direction can help the research and practice communities use the error taxonomies throughout the software process in order to ensure a more complete verification process.

## 7. 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. Gary Bradshaw, a human cognition and psychology expert, for providing information on relevant databases and useful feedback on our research. Finally, we thank Doris Carver and Guilherme Travassos for reviewing early drafts of this paper.

## REFERENCES

1. *Software Engineering Laboratory: Software Measurement Guidebook*. 1994, NASA/GSFC Software Engineering Laboratory.
2. Ackerman, A.F., Buchwald, L.S., and Lewski, F.H., "Software Inspections: An Effective Verification Process." *IEEE Software*, 1989. **6**(3): 31-36.
3. Aurum, A., Petersson, H., and Wohlin, C., "State-of-the-art: Software Inspections after 25 Years." *Journal of Software Testing Verification and Reliability*, 2002. **12**(3): 133-154.
4. Bailey, B.P. and Konstan, J.A., "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*, 2006. **22**(4): 685-708.
5. Barnard, J., Emam, K., and Zubrow, D. "Getting More Out of Your Inspection Data: Using Capture-Recapture Models for the Reinspection Decision". In *Proceedings of the European SEPG Conference*. 2002. Amsterdam: IEEE Press: 1-25
6. Basili, V. and Green, S., "Software Process Evolution at the SEL." *IEEE Software*, 1994. **11**(2): 58-66.
7. Basili, V.R. and Weiss, D.M. "Evaluation of a Software Requirements Document by Analysis of Change Data". In *Proceedings of the 5th International Conference on Software Engineering*. 1981. San Diego, CA: IEEE Press: 314-323
8. Basili, V.R. and Perricone, B.T., "Software Errors and Complexity: An Empirical Investigation." *Communications of the ACM*, 1984. **27**(1): 42-52.
9. Basili, V.R., Katz, E.E., Panlilio-Yap, N.N., Ramsey, C.L., and Chang, S., "Characterization of an Ada Software Development." *IEEE Computer*, 1985. **18**(9).
10. Basili, V.R. and Rombach, H.D. "Tailoring the Software Process to Project Goals and Environments". In *Proceedings of 9th International Conference in Software Engineering*. 1987. California, United States IEEE Press: 345-357
11. Basili, V.R. and Rombach, H.D., "The TAME Project: Towards Improvement-Oriented Software Environments." *IEEE Transactions on Software Engineering*, 1988. **14**(6): 758-772.
12. Basili, V.R., "Evolving and Packaging Reading Techniques." *Journal of Systems and Software*, 1997. **38**(1): 3-12.
13. Basu, S. and Ebrahimi, N. "Estimating the Number of Undetected Errors: Bayesian Model Selection". In *Proceedings of the 9th International Symposium on Software Reliability Engineering*. 1998. Paderborn, Germany: IEEE Computer Society: 22-31
14. Beecham, S., Hall, T., Britton, C., Cottee, M., and Rainer, A., "Using an Expert Panel to Validate a Requirements Process Improvement Model." *The Journal Of Systems and Software*, 2005. **76**(3): 251-275.
15. Bell, T.E. and Thayer, T.A. "Software Requirements: Are They Really a Problem?" In *Proceedings of 2nd International Conference on Software Engineering*. 1976. Los Alamitos, CA: IEEE Computer Society Press: 61-68
16. Benson, J.P. and Saib, S.H. "A Software Quality Assurance Experiment". In *Proceedings of the software quality assurance workshop on Functional and performance issues*. 1978: ACM Press 87-91

17. Berling, T. and Thelin, T. "A Case Study of Reading Techniques in a Software Company". In *Proceedings of the 2004 International Symposium on Empirical Software Engineering (ISESE'04)*. 2004: IEEE Computer Society: 229-238
18. Bernardez, B., Genero, M., Duran, A., and Toro, M. "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)*. 2004: IEEE Computer Society: 257-268
19. Bhandari, I., Halliday, M., Tarver, E., Brown, D., Chaar, J., and Chillarege, R., "A Case Study of Software Process Improvement During Development." *IEEE Transactions on Software Engineering*, 1993. **19**(12): 1157-1170.
20. Bhandari, I., Halliday, M.J., Chaar, J., Chillarege, R., Jones, K., Atkinson, J.S., Lepori-Costello, C., Jasper, P.Y., Tarver, E.D., Lewis, C.C., and Yonezawa, M., "In Process Improvement Through Defect Data Interpretation." *IBM Systems Journal*, 1994. **33**(1): 182-214.
21. Biffel, S., Freimut, B., and Laitenberger, O. "Investigating the Cost-Effectiveness of Reinspections in Software Development". In *Proceedings of the 23rd International Conference on Software Engineering*. 2001. Toronto, Ontario, Canada: IEEE Computer Society: 155-164
22. Blackburn, M.R., Busser, R., and Nauman, A. "Eliminating Requirement Defects and Automating Test". In *Test Computer Software Conference*. 2001. IEEE Computer Society: 25-34
23. Blavier, A., Rouy, E., Nyssen, A.S., and Keyser, V., "Prospective Issues for Error Detection." *Journal of Ergonomics*, 2005. **48**(7): 758-781.
24. Boehm, B. and Basili, V.R., "Software Defect Reduction Top 10 List." *IEEE Computer*, 2001. **34**(1): 135-137.
25. Bove, T., *Development and Validation of Human Error Management Taxonomy in Air Traffic Control*, in *Risø National Laboratory & University of Roskilde*. 2002. p. 234.
26. Browne, G.J. and Ramesh, V., "Improving Information Requirements Determination: A Cognitive Perspective." *Journal of Information and Management*, 2002. **39**(8): 625-645.
27. Brykczynski, B., "A Survey of Software Inspection Checklists." *ACM SIGSOFT Software Engineering Notes*, 1999. **24**(1): 82-89.
28. Buse, D.K. and Johnson, C.W. "Identification and Analysis of Incidents in Complex, Medical Environments". In *Proceedings of the First Workshop on Human Error and Clinical Systems*. 1999. Glasgow, Scotland: Cambridge University Press:
29. Cacciabue, P.C., "A Methodology of Human Factors Analysis for Systems Engineering: Theory and Applications." *IEEE Transactions on System, Man And Cybernetics-Part A: Systems And Humans*, 1997. **27**(3): 325-329.
30. Card, D.N., "Learning from Our Mistakes with Defect Causal Analysis." *IEEE Software*, 1998. **15**(1): 56-63.
31. Cheng, B. and Jeffrey, R. "Comparing Inspection Strategies for Software Requirement Inspections". In *Proceedings of the 1996 Australian Software Engineering Conference*. 1996. Melbourne, Australia: IEEE Computer Society: 203-211
32. Chernak, Y., "A Statistical Approach to the Inspection Checklist Formal Synthesis and Improvement." *IEEE Transactions on Software Engineering*, 1996. **22**(12): 866-874.

33. Chillarege, R., Bhandari, I., Chaar, J., Halliday, M., Moebus, D., Ray, B., and Wong, M.Y., "Orthogonal Defect Classification- A Concept for In-process Measurement." *IEEE Transactions on Software Engineering*, 1992. **18**(11): 943-956.
34. Ciolkowski, M., Laitenberger, O., and Biffi, S., "Software Reviews: The State of the Practice." *IEEE Software*, 2003. **20**(6): 46-51.
35. Clark, B. and Zubrow, D. "How Good Is the Software: A Review of Defect Prediction Techniques". In *Software Engineering Symposium*. 2001. Pittsburgh, PA: IEEE Computer Press: 1-35
36. Cristopher, W.J., "The Cost of Errors in Software Development: Evidence from Industry." *The Journal of System and Software*, 2002. **62**(1): 1-9.
37. Debou, C. and Combelles, A.K., "Linking Software Process Improvement to Business Strategies: Experiences from Industry." *Journal of Software Process: Improvement and Practice*, 2000. **5**(1): 55-64.
38. Dekker, S.W.A., "Illusions of Explanation : A Critical Essay on Error Classification." *The International Journal of Aviation Psychology*, 2003. **13**(2): 95-106.
39. Dhillon, B.S. and Liu, Y., "Human Error in Maintenance: A Review." *Journal of Quality in Maintenance Engineering*, 2006. **12**(1): 21-36.
40. Duncan, I.M.M. and Robson, D.J., "An Exploratory Study of Common Coding Faults in C Programs." *C Traps and Pitfalls*, 1996. **8**(4): 251-256.
41. Dyba, T., *Experiences of Undertaking Systematic Reviews*. 2005, SINTEF ICT: Queensland.
42. Emam, K., Laitenberger, O., and Harbich, T., "The Application of Subjective Estimates of Effectiveness to Controlling Software Inspections." *The Journal Of Systems and Software*, 2000. **54**(2): 119-136.
43. Endres, A., "An Analysis of Errors and Their Causes in System Programs." *IEEE Transactions on Software Engineering*, 1975. **1**(2): 140-149.
44. Endsley, M.R. "Situation Awareness and Human Error: Designing to Support Human Performance". In *Proceedings of the High Consequence Systems Surety Confrence*. 1999. Albuquerque, NM: SA Technologies: 2-9
45. Fagan, M.E., "Design and Code Inspections to Reduce Errors in Program Development." *IBM Systems Journal*, 1976. **15**(3): 182-211.
46. Fenton, N.E. and Neil, M., "A Critique of Software Defect Prediction Models." *IEEE Transactions on Software Engineering*, 1999. **25**(5): 675-689.
47. Fitts, P.M. and Jones, R.E. "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*. 1961: New York: Dover Publications Inc.: 332-358
48. Florac, W.A., *Software Quality Measurement: A Framework for Counting Problems and Defects*. 1992, Carnegie Mellon Software Engineering Institute: Pittsburgh, PA.
49. Fredericks, M. and Basili, V., *Using Defect Tracking and Analysis to Improve Software Quality*. 1998, The Data & Analysis Center for Software (DACs)
50. Freimut, B., Denger, C., and Ketterer, M. "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*. 2005: IEEE Press:
51. Fujii, M.S., "A Comparison of Software Assurance Methods." *ACM SIGMETRICS Performance Evaluation Review* 1978 **7** (3-4): 27 - 32

52. Fusaro, P., Lanubile, F., and Visaggio., G., "A Replicated Experiment to Assess Requirements Inspection Techniques." *Journal of Empirical Software Engineering*, 1997. **2**(1): 39-57.
53. Gaitros, D.A., "Common Errors in Large Software Development Projects." *The Journal of Defense Software Engineering*, 2004. **12**(6): 21-25.
54. Galliers, J., Minocha, S., and Sutcliffe, A., "A Causal Model of Human Error for Safety Critical User Interface Design." *ACM Transactions on Computer-Human Interaction*, 1998. **5**(3): 756-769.
55. Galliers, J., Sutcliffe, A., and Minocha, S., "An Impact Analysis Method for Safety-Critical User Interface Design." *ACM Transactions on Computer-Human Interaction*, 1999. **6**(4): 341-369.
56. Graboswki, M. and Roberts, K.H., "Human and Organizational Error in Large Scale Systems." *IEEE Transactions on System, Man And Cybernetics-Part A: Systems And Humans*, 1996. **26**(1): 2-16.
57. Grady, R.B., "Software Failure Analysis for High-Return Process Improvement." *Hewlett-Packard Journal*, 1996. **47**(4): 15-24.
58. Gray, W.D., "The Nature and Processing of Errors in Interactive Behavior." *Journal of Cognitive Science*, 2000. **24**(2): 205-248.
59. Hall, T., Beecham, S., and Rainer, A., "Requirement Problems in Twelve Software Companies: An Empirical Analysis." *IEE Proceedings Software*, 2002. **149**(5): 153-160.
60. Hayes, J.H. "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*. 2003: IEEE Computer Society: 49- 59
61. Hayes, J.H., Holbrook, E.A., Raphael, I., and Pruett, D.M. "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)*. 2005. Chicago: IEEE Computer Society:
62. Henningsson, K. and Wohlin, C. "Assuring Fault Classification Agreement- An Empirical Evaluation". In *Proceedings of the 2004 International Symposium on Empirical Software Engineering (ISESE'04)*. 2004: IEEE Computer Society: 95-104
63. Hobbs, A. and Williamson, A., "Skills, Rules and Knowledge in Aircraft Maintenance: Errors in Context." *Journal of Ergonomics*, 2002. **45**(4): 290-308.
64. Hollnagel, E., *Human Reliability Analysis: Context and Control*. 1994, London: Academic Press.
65. Huang, J.C., Chang, C.K., and Christensen, M., "Event-Based Traceability for Managing Evolutionary Change." *IEEE Transactions on Software Engineering*, 2003. **29**(9): 796-804.
66. Humphrey, W.S., "Using A Defined and Measured Personal Software Process." *IEEE Software*, 1996. **13**(3): 77-88.
67. Issac, A., Shorrock, S.T., Kennedy, R., Kirwan, B., Andersen, H., and Bove, T., *The Human Error in ATM Technique (HERA-JANUS)*. 2002, European Air Traffic Management. p. 1-94.
68. Jacobs, J., Moll, J.V., Krause, P., Kusters, R., Trienekens, J., and Brombacher, A., "Exploring Defect Causes in Products Developed by Virtual Teams." *Journal of Information and Software Technology*, 2005. **47**(6): 399-410.

69. Jaffe, M.S., Leveson, N.G., Heimdahl, M.P.E., and Melhart, B.E., "Software Requirements Analysis for Real-Time Process Control Systems." *IEEE Transactions on Software Engineering*, 1991. **17**(3): 241-258.
70. Johnson, C., "Forensic Software Engineering: Are Software Failures Symptomatic of Software Problems?" *Journal of Safety Science*, 2002. **40**(9): 835-847.
71. Jorgensen, M. and Shepperd, M., "A Systematic Review of Software Development Cost Estimation Studies." *IEEE Transactions on Software Engineering*, 2007. **33**(1): 33-53.
72. Joung, W. and Hesketh, B., "Using "War Stories" to Train for Adaptive Performance: Is it Better to Learn from Errors or Success?" *Journal of Applied Psychology: An International Review*, 2006. **55**(2): 282-302.
73. Kan, S.H., Basili, V.R., and Shapiro, L.N., "Software Quality: An Overview From The Perspective Of Total Quality Management." *IBM Systems Journal*, 1994. **33**(1): 4-19.
74. Kantorowitz, E., Arzi, L., and Gutmann, A., "The Performance of the N-Fold Requirement Inspection Method." *Requirement Engineering Journal*, 1997. **2**(3): 152-164.
75. Kelly, D. and Shepard, T. "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 2000*. Mississauga, Ontario, Canada IBM Press
76. Kirner, T.G. and Abib, J.C. "Inspection of Software Requirements Specification Documents: A Pilot Study". In *Proceedings of the 15th annual international conference on Computer documentation*. 1997. Salt Lake City, Utah, United States IEEE Press: 161 - 171
77. Kitchenham, B., *Procedures for Performing Systematic Reviews*. 2004, Department of Computer Science, Keele University and National ICT, Australia Ltd. p. 33.
78. Knight, J.C. and Myers, A.E., "An Improved Inspection Technique." *Communications of the ACM*, 1993. **36**(11): 50-69.
79. Ko, A.J. and Myers, B.A. "Development and Evaluation of a Model of Programming Errors". In *Proceedings of IEEE Symposium on Human Centric Computing Languages and Environments*. 2003: IEEE Computer Society: 7-14
80. Ko, A.J. and Myers, B.A., "A Framework and Methodology for Studying the Causes of Software Errors in Programming Systems." *Journal of Visual Languages and Computing*, 2005. **16**(2): 41-84.
81. Krogstie, J., "Integrating the Understanding of Quality in Requirements Specification and Conceptual Modeling." *ACM SIGSOFT Software Engineering Notes*, 1998. **23**(1): 86-91.
82. Laitenberger, O. and Atkinson, C. "Generalizing Perspective-based Inspection to Handle Object-Oriented Development Artifacts". In *International Conference on Software Engineering*. 1999. Los Angeles, CA, USA: IEEE Computer Society Press: 494-503
83. Laitenberger, O., Atkinson, C., Schlich, M., and Emam, K.E., "An Experimental Comparison of Reading Techniques for Defect Detection in UML Design Documents." *The Journal Of Systems and Software*, 2000. **53**(2): 183-204.
84. Laitenberger, O. and DeBaud, J.M., "An Encompassing Life Cycle Centric Survey of Software Inspection." *The Journal Of Systems and Software*, 2000. **50**(1): 5-31.

85. Lanubile, F. and Visaggio, G., *Assessing Defect Detection Methods for Software Requirement Inspection Through External Replication*. 1996, Dept. of Informatica, University of Bari.
86. Lanubile, F., Shull, F., and Basili, V.R. "Experimenting with Error Abstraction in Requirements Documents". In *Proceedings of the 5th International Symposium on Software Metrics (METRIC'98)*. 1998. Bethesda, MD, USA: IEEE Computer Society: 114-121
87. Lawrence, C.P. and Kosuke, I., "Design Error Classification and Knowledge Management." *Journal of Knowledge Management Practice*, 2004. **10**(9): 72-81.
88. Leszak, M., Perry, D.E., and Stoll, D. "A Case Study in Root Cause Defect Analysis". In *Proceedings of the 22nd International Conference on Software Engineering*. 2000. Limerick, Ireland ACM Press 428 - 437
89. Linger, R.C., Mills, H.D., and Witt, B.I., *Structured Programming: Theory and Practice*. 1979. : Addison-Wesley Publishing Company.
90. Lutz, R.R., Wong, K., and Johnny, S. "Constraint Checking During Error Recovery". In *Proceedings of the NASA Technology 2002 Conference*. 1992: IEEE Computer Society: 142-153
91. Lutz, R.R. "Analyzing Software Requirements Errors in Safety-Critical, Embedded Systems". In *Proceedings of the IEEE International Symposium on Requirements Engineering*. 1993. San Diego, CA, USA: IEEE Computer Society Press: 126-133
92. Lutz, R.R., "Targeting Safety-Related Errors During Software Requirements Analysis." *The Journal of systems and software* 1996. **34**(3): 223-230
93. Lutz, R.R., "Requirements Analysis Using Forward and Backward Search." *Annals of Software Engineering*, 1997. **3**(1): 459-475.
94. Martin, J. and Tsai, W.T., "N-Fold Inspection: A Requirement Analysis Technique." *Communications of the ACM*, 1990. **33**(2): 225-232.
95. Masuck, C., "Incorporating A Fault Categorization and Analysis Process in the Software Build Cycle." *Journal of Computing Sciences in Colleges* 2005. **20**(5): 239 - 248
96. Mays, R.G., Jones, C.L., Holloway, G.J., and Studinski, D.P., "Experiences with Defect Prevention." *IBM Systems Journal*, 1990. **29**(1): 4 - 32
97. McGarry, F., Page, G., Basili, V., and Zelkowitz, M., *Software Process Improvement in the NASA Software Engineering Laboratory*. 1994, Carnegie Mellon Software Engineering Institute (SEI).
98. Mendis, K.S., "Quantifying Software Quality." *Annual Quality Congress Transaction*, 1981. **35**(4): 11-18
99. Mian, P., Conte, T., Natatli, A., Biolchini, J., and Travassos, G., *A Systematic Review Process for Software Engineering*. 2005, Computer Science Department, Brazil.
100. Miller, J., Wood, M., and Roper, M., "Further Experiences with Scenarios and Checklists." *Journal of Empirical Software Engineering*, 1998. **3**(1): 37-64.
101. Miller, J., "Estimating the Number of Remaining Defects after Inspection." *Journal of Software Testing, Verification and Reliability*, 1999. **9**(3): 167-189.
102. Mohri, Y. and Kikuno, T. "Fault Analysis Based on Fault Reporting in JSP Software Development". In *Proceedings of the 15th Annual International Computer Software and Applications Conference* 1991: IEEE Computer Society: 591-596

103. Munson, J.C. and Nikora, A.P. "Toward A Quantifiable Definition of Software Faults ". In *Proceedings of the 13th International Symposium on Software Reliability Engineering*. 2002: IEEE Computer Society: 388- 395
104. Nachiappan, N., Laurie, W., John, H., Will, S., and Mladen, V. "Preliminary Results on Using Static Analysis Tools for Software Inspection". In *Proceedings of the 15th International Symposium on Software Reliability Engineering (ISSRE'04)* 2004: IEEE Press: 429- 439
105. Nakashima, T., Oyama, M., Hisada, H., and Ishii, N., "Analysis of Software Bug Causes and Its Prevention." *Journal of Information and Software Technology*, 1999. **41**(15): 1059-1068.
106. Nelms, C.R. "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'*, 1997 1997. Orlando, FL, USA: IEEE Press: 7-12
107. Nieves, J.M. and Sage, A.P., "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*, 1998. **28**(6): 742-762.
108. Norman, D., *The Psychology of Every Day Things*. 1988, New York: Basic Books.
109. Norman, D.A., "Steps Towards a Cognitive Engineering: Design Rules Based on Analyses of Human Error." *Communications of the ACM*, 1981. **26**(4): 254-258.
110. Norman, D.A., "Design Rules Based on Analyses of Human Error." *Communications of the ACM*, 1983. **26**(4): 254-258.
111. Oliveira, K.M., Zlot, F., Rocha, A.R., Travassos, G.H., Galotta, C., and Menezes, C.S., "Domain-Oriented Software Development Environment." *The Journal Of Systems and Software*, 2004. **72**(2): 145-161.
112. Paterno, F. and Santoro, C., "Preventing User Errors by Systematic Analysis of Deviations from the System Task Model." *International Journal of Human-Computer Studies*, 2002. **56**(2): 225-245.
113. Patrick, F., David, L., Melinda, M., and Andy, P. "Tree-Based Methods for Classifying Software Failures". In *Proceedings of the 15th International Symposium on Software Reliability Engineering (ISSRE'04)*. 2004: IEEE Press: 451- 462
114. Porter, A.A., Votta, L.G., and Basili, V.R., "Comparing Detection Methods for Software Requirements Inspections: A Replicated Experiment." *IEEE Transactions on Software Engineering*, 1995. **21**(6): 563-575.
115. Porter, A.A. and Votta, L.G., "What Makes Inspections Work?" *IEEE Software*, 1997. **14**(6): 99-102.
116. Prechelt, L., "Accelerating Learning from Experience: AVOIDING Defects Faster." *IEEE Software*, 2001. **18**(6): 56-61.
117. Reason, J., *Human Error*. 1990, Cambridge, USA: Cambridge University Press.
118. Reinach, S. and Viale, A., "Application of a Human Error Framework to Conduct Train Accident/Incident Investigations." *Journal of Accident Analysis and Prevention*, 2006. **38**(2): 396-406.
119. Robillard, P.N., "The Role of Knowledge in Software Development." *Communications of the ACM*, 1999. **42**(1): 87-92.

120. Robinson, W.N. and Pawlowski, S.D., "Managing Requirement Inconsistency with Development Goal Monitors." *IEEE Transactions on Software Engineering*, 1999. **25**(6): 816-835.
121. Sabaliauskaite, G., Kusumoto, S., and Inoue, K., "Assessing Defect Detection Performance of Interacting teams in Object-Oriented Design Inspections." *Journal of Information and Software Technology*, 2004. **46**(13): 875-886.
122. Sakthivel, S., "A Survey of Requirement Verification Techniques." *Journal of Information Technology*, 1991. **6**(2): 68-79.
123. Sandahl, K., Blomkvist, O., Karlsson, J., Krysanter, C., Lindvall, M., and Ohlsson, N., "An Extended Replication of an Experiment for Assessing Methods for Software Requirements Inspections." *Journal of Empirical Software Engineering*, 1998. **3**(4): 327-354.
124. Sasao, K. and Reason, J., "Team Errors: Definition and Taxonomy." *Journal of Reliability Engineering and System Safety*, 1999. **65**(1): 1-9.
125. Sawyer, P., Sommerville, I., and Viller, S., "Capturing the Benefits of Requirement Engineering." *IEEE Software*, 1999. **16**(2): 78-85.
126. Schneider, G.M., Martin, J., and Tsai, W.T., "An Experimental Study of Fault Detection in User Requirements Documents." *ACM Transactions on Software Engineering and Methodology*, 1992. **1**(2): 188-204.
127. Senders, L.W. and Moray, N.P., *Human Error: Cause, Prediction, and Reduction*. 1991, Hillsdale, NJ: Lawrence Erlbaum.
128. Shappell, S.A. and Weigmann, D.A., "A Human Error Approach to Accident Investigation: The Taxonomy of Unsafe Operations." *The International Journal of Aviation Psychology*, 1997. **7**(4): 269-291.
129. Shen, W., Guizani, M., Yang, Z., Compton, K., and Huggins, J. "Execution of A Requirement Model in Software Development". In *Proceedings of the 13th International Conference on Intelligent & Adaptive Systems and Software Engineering*. 2004. Nice, France: IEEE Press: 203-208
130. Shorrock, S.T. and Kirwan, B., "Development and Application of a Human Error Identification Tool for Air Traffic Control." *Journal of Applied Ergonomics*, 2002. **33**(4): 319-336.
131. Shull, F., Rus, I., and Basili, V., "How Perspective Based Reading Can Improve Requirement Inspection." *IEEE Computer*, 2000. **33**(7): 73-79.
132. Smith, J., "The 40 Root Causes of Troubled IT Projects." *Journal of IEEE Computer and Control Engineering*, 2002. **13**(3): 109-112.
133. Stanton, N.A. and Stevenage, S.V., "Learning to Predict Human Error: Issues of Acceptability, Reliability and Validity." *Journal of Ergonomics*, 1998. **41**(11): 1737-1756.
134. Stutzke, M.A. and Smidts, C.S., "A Stochastic Model of Fault Introduction & Removal During Software Development." *IEEE Transactions on Reliability*, 2001. **50**(20): 184-193.
135. Sutcliffe, A. and Rugg, G., "A Taxonomy of Error Types for Failure Analysis and Risk Assessment." *International Journal of Human-Computer Interaction*, 1998. **10**(4): 381-405.

136. Swain, A. and Guttman, H., *Handbook of Human Reliability Analysis with Emphasis on Nuclear Power Plant Applications*. 1983, Nuclear Regulatory Commission: Washington, DC.
137. Thelin, T. and Runeson, P., "Robust Estimation of Fault Content with Capture-Recapture and Detection Profile Estimators." *The Journal Of Systems and Software*, 2000. **52**(2): 139-148.
138. Thelin, T., Runeson, P., Wohlin, C., Olsson, T., and Andersson, C. "Team Based Fault Content Estimation in the Software Inspection Process". In *Proceedings of the 26th International Conference on Software Engineering (ICSE'04)*. 2004: IEEE Computer Society: 263- 272
139. Thelin, T., Runeson, P., Wohlin, C., Olsson, T., and Andersson, C., "Evaluation of Usage Based Reading- Conclusion after Three Experiments." *Journal of Empirical Software Engineering*, 2004. **9**(1-2): 77-110.
140. Travassos, G.H., Shull, F., Fredericks, M., and Basili, V.R. "Detecting Defects in Object Oriented Designs: Using Reading Techniques to Increase Software Quality". In *Proceedings of Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'99)*. 1999. Denver, Colorado: IEEE Press:
141. Trevor, C., Jim, S., Judith, C., and Brain, K., *Human Error in Software Generation Process*. 1994, University of Technology, Loughborough, England
142. Venkatasubramaniam, V., Rengaswamy, R., and Kavuri, S.K., "A Review of Process Fault Detection and Diagnosis Part 2: Qualitative Models and Search Strategies." *Journal of Computers and Chemical Engineering*, 2003. **27**(2): 313-326.
143. Venkatasubramaniam, V., Rengaswamy, R., Yin, K., and Kavuri, S.K., "A Review of Fault Detection and Diagnosis Part 1: Quantitative Model-Based Methods." *Journal of Computers and Chemical Engineering*, 2003. **27**(2): 293-311.
144. Viller, S., Bowers, J., and Rodden, T., *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*. 1997, Computing Department, Lancaster University: Lancaster.
145. Walia, G.S., Carver, J., and Philip, T. "Requirement Error Abstraction and Classification: An Empirical Study". In *IEEE Symposium on Empirical Software Engineering*. 2006. Brazil: ACM Press: 336-345
146. Weiss, D.M., "Evaluating Software Development by Error Analysis: The Data from Architecture Research Facility." *Journal of Systems and Software*, 1979. **4**(4): 289-300.
147. Yamamura, T., Yata, K., Yasushi, T., and Yamaguchi, H. "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*,. 1989. Dallas, TX, USA: IEEE Press: 795-800
148. Zage, D. and Zage, W. "An Analysis of the Fault Correction Process in a Large-Scale SDL Production Model". In *Proceedings of the 25th International Conference on Software Engineering*. 2003. Portland, Oregon: IEEE Computer Society: 570 - 577
149. Zhang, X. and Pham, H., "An Analysis of Factors Affecting Software Reliability." *The Journal Of Systems and Software*, 2000. **50**(1): 43-56.

## APPENDIX A INITIAL STUDIES SELECTION

Table 10 summarizes the results of the search process and indicates which papers were included in the review. The columns in the table provide the following information:

- *Database* – the database from Table 2 that was searched;
- *Search String* – the string from Table 3 that was used;
- *# of hits* - the total number of papers returned by the search;
- *Title* – the number of papers from previous column that were still candidates for inclusion after a review of the title;
- *Abstract & Keyword* – the number of papers from the previous column that were still candidates for inclusion after reviewing the abstract and keywords;
- *Papers Selected* – the final list of papers that were included in the review after reviewing the entire paper.

For example, in the first row of Table 10, search string #1 was executed on the IEEEXplore database and the best 500 results were returned; out of those 500 papers, 120 papers were selected based on their title. The abstract and keywords for those 120 papers were reviewed, further narrowing down the list to 21 papers. Finally, those 21 papers were read and 12 papers were determined to be relevant.

**Table 10 - Papers Selected**

Data Base	Search String	# of hits	Title	Abstract & Keywords	Papers Selected
IEEEExplore:	1	Found 178380 of 1351415 documents, Best 500 shown.	120	21	[6, 18, 19, 30, 32, 50, 66, 88, 125, 131, 134, 148]
IEEEExplore:	2	Found 181692 of 1351415 documents, Best 500 shown.	115	12	[65]
IEEEExplore:	3	Found 83348 of 1351415 documents, Best 500 shown.	223	8	[29, 56, 62]
IEEEExplore:	4	Found 365025 of 1351415 documents, Best 500 shown.	185	13	[2, 13, 17, 86, 114, 115]
IEEEExplore:	5	Found 167 of 1351415 documents.	54	7	[79, 107, 147]
IEEEExplore:	6	Found 15 of 1351415 documents.	7	2	-----
IEEEExplore:	7	Found 35282 of 1351415 documents, Best 500 shown.	95	11	[106, 132]
<b>Total References Selected -- 27</b>					
INSPEC	1	<b>692</b> results shown	<b>130</b>	<b>8</b>	[20, 45, 73]
INSPEC	2	<b>303</b> results shown	<b>100</b>	<b>6</b>	[26, 122, 129]
INSPEC	3	<b>565</b> results shown	<b>120</b>	<b>8</b>	[58, 104, 134]
INSPEC	4	<b>2178</b> results shown	<b>80</b>	<b>8</b>	[101, 113, 121, 139]
INSPEC	5	<b>118</b> results shown	<b>79</b>	<b>5</b>	[58]
INSPEC	6	<b>118</b> results shown	<b>100</b>	<b>0</b>	-----
INSPEC	7	<b>198</b> results shown	<b>120</b>	<b>8</b>	[142, 143]
<b>Total References Selected -- 16</b>					
ACM Portal	1	Found <b>48,623</b> of <b>177,263</b> , Best <b>200</b> shown	<b>110</b>	<b>15</b>	[16, 21, 51, 81, 95, 98, 138]
ACM Portal	2	Found <b>31,535</b> of <b>128</b> searched out of <b>156,829</b> , Best <b>200</b> shown	<b>94</b>	<b>13</b>	[8, 126]
ACM Portal	3	Found <b>28,184</b> of <b>31,535</b> out of <b>156,829</b> , Best <b>200</b> shown	<b>120</b>	<b>6</b>	[27, 43, 75]
ACM Portal	4	Found <b>6,996</b> of <b>22,609</b> searched out of <b>156,829</b> , Best <b>200</b> shown.	<b>105</b>	<b>14</b>	[75, 76, 82, 94, 140]
ACM Portal	5	Found <b>42,722</b> of <b>16,996</b> searched out of <b>156,829</b> , Best <b>200</b> Shown.	<b>167</b>	<b>6</b>	[54, 109, 110]

ACM Portal	6	Found <b>7,445</b> of <b>42,722</b> searched out of <b>156,829</b> , Best <b>200</b> shown.	<b>134</b>	<b>0</b>	----- ----- -----
ACM Portal	7	Found <b>1,234</b> of <b>2,609</b> searched out of <b>156,829</b> , Best <b>200</b> shown.	<b>42</b>	<b>2</b>	----- ----- -----
<b>Total References Selected -- 20</b>					
SCIRUS (Elsevier)	1	<b>411,902</b> journal results, narrowed to <b>2172 (software engineering)</b> , narrowed further to <b>385 (software quality)</b> .	<b>212</b>	<b>15</b>	[42, 68, 70, 84, 103, 105, 111, 137]
SCIRUS (Elsevier)	2	<b>6,352</b> journal results, narrowed to <b>1205 (software engineering)</b> ,	<b>313</b>	<b>7</b>	[14, 149]
SCIRUS (Elsevier)	3	<b>7,237</b> journals results; narrowed to <b>553</b> journal results (software engineering).	<b>212</b>	<b>7</b>	[40]
SCIRUS (Elsevier)	4	<b>15,909</b> journal results narrowed to <b>282</b> journal results (software engineering).	<b>84</b>	<b>11</b>	[52, 83, 85]
SCIRUS (Elsevier)	5	<b>2,014</b> journal results.	<b>221</b>	<b>6</b>	[39, 112]
SCIRUS (Elsevier)	6	<b>10,711</b> journal results narrowed to <b>129</b> journals (software development process).	<b>75</b>	<b>7</b>	[144]
SCIRUS (Elsevier)	7	<b>3, 231</b> journal results narrowed to <b>385</b> journals (software engineering).	<b>103</b>	<b>10</b>	[80]
<b>Total References Selected -- 18</b>					
Google scholar	1	<b>34,600</b> English articles searched only in engineering, computer science between years 1997-2006.	<b>847</b>	<b>25</b>	[10, 12, 24, 46]
Google scholar	2	<b>13,300</b> English articles searched only in engineering, computer science between years 1997-2006.	<b>365</b>	<b>21</b>	[22, 37, 59, 120]
Google scholar	3	<b>33,900</b> English articles searched only in engineering, computer science between years 1997-2006.	<b>651</b>	<b>31</b>	[60]
Google scholar	4	<b>11,700</b> English articles searched only in engineering, computer science between years 1997-2006.	<b>542</b>	<b>34</b>	[3, 34, 74]
Google scholar	5	<b>23,390</b> English articles searched only in engineering, computer science between years 1997-2006.	<b>1012</b>	<b>45</b>	[28, 44, 124, 130, 133]
Google scholar	6	<b>3,000</b> English articles searched only in engineering, computer science between years 1997-2006.	<b>53</b>	<b>5</b>	[119]
Google scholar	7	<b>2,000</b> English articles searched only in engineering, computer science between years 1997-2006.	<b>53</b>	<b>6</b>	[49]
<b>Total References Selected -- 19</b>					
SEI technical report website	1	Best <b>100</b> of about <b>465</b> articles shown	<b>85</b>	<b>5</b>	[35, 48, 61, 97]

SEI technical report website	2	Best <b>332</b> of about <b>940</b> articles shown.	<b>132</b>	<b>8</b>	-----
SEI technical report website	3	Best <b>93</b> of about <b>175</b> results shown.	<b>89</b>	<b>4</b>	-----
SEI technical report website	4	Best <b>158</b> of about <b>329</b> results shown.	<b>75</b>	<b>7</b>	[5]
SEI technical report website	5	Best <b>130</b> of about <b>324</b> results shown.	<b>54</b>	<b>0</b>	-----
SEI technical report website	7	Best <b>113</b> of about <b>288</b> results shown.	<b>69</b>	<b>5</b>	-----
<b>Total References Selected -- 5</b>					
Springer Link	1	<b>235</b> results shown.	<b>54</b>	<b>3</b>	-----
Springer Link	2	<b>125</b> result shown.	<b>34</b>	<b>2</b>	-----
Springer Link	3	<b>27</b> results shown.	<b>16</b>	<b>1</b>	-----
Springer Link	4	<b>123</b> results shown.	<b>67</b>	<b>7</b>	[52, 100, 123]
Springer Link	5	<b>343</b> results shown.	<b>43</b>	<b>12</b>	-----
Springer Link	6	<b>35</b> results shown.	<b>12</b>	<b>4</b>	-----
Springer Link	7	<b>96</b> results shown.	<b>33</b>	<b>6</b>	-----
<b>Total References Selected -- 3</b>					
PsychINFO (EBSCO)	Human errors	<b>4426</b> results shown.		<b>17</b>	[4, 63, 72, 133]
PsychINFO (EBSCO)	Error classification	<b>1096</b> results shown.		<b>12</b>	[23, 38, 118, 128]
PsychINFO (EBSCO)	6	<b>64</b> results shown.		<b>0</b>	-----
<b>Total References Selected -- 8</b>					
Science Citation Index	James Reason	<b>62</b> results shown	<b>32</b>	<b>7</b>	-----
Books	[64, 108, 117, 127]				
<b>Total References Selected -- 4</b>					
Reference Lists from Primary Studies and other review studies	[10, 15, 31, 53, 67, 69, 78, 90-93, 96, 116]				
<b>Total References Selected -- 13</b>					

## APPENDIX B DATA EXTRACTION FORMS

This appendix describes the data extraction forms that were used for recording useful information from the selected papers. Two different data extraction forms were used for this purposed as described in Table 11 and Table 12.

Table 11 describes the data items that were common to all the research papers (for example study identifier, study aims, study design, study findings, relevance etc). So, for each selected paper, all the rows describing different data items in Table 11 were filled.

An additional form was created as described in Table 12. Since, different search strings were executed pertaining to different foci of the review questions. So, to extract information specific to selected papers from a particular search string, part(s) of the data items in Table 12 was filled. Table 12 describes data items for different search focus: quality improvement approach, requirement errors, error-fault-defect taxonomies, software inspections, and human errors.

So, for each paper all the information in Table 11 was recorded, but only the information relevant to particular paper depending on the search focus was recorded in Table 12.

**Table 11 - Data Items Extracted from All Papers**

<b>Data item</b>	<b>Description</b>
Identifier	Unique identifier for the paper (same as the reference number)
Bibliographic data	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, 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.
Other	Any issue of importance not covered by the above.
Relevance	Relevance to the current research

**Table 12 - Data Items Described for each Search Focus**

<b>Search Focus</b>	<b>Data item</b>	<b>Description</b>
<i>Quality Improvement Approach</i>	Focus or process	Focus of the quality improvement approach and the process/method used to improve quality
	Benefits	Any benefits applying the approach identified
	Limitations	Any limitations or problems identified
	Evidence	The nature of evidence presented
	Error focus	Yes or No; and if “Yes”, how does it relate to the current research
<i>Requirement Errors</i>	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
<i>Error-fault-defect taxonomies</i>	Focus	The focus of the taxonomy (i.e. error, fault, or failure)
	Error Focus	Yes or No; if “Yes”, how does it relate to the current research questions
	Requirement phase	Yes or No (whether it is applied in requirement phase)
	Benefits	Benefits of the taxonomy
	Limitation	Limitations of the taxonomy
	Evidence	The nature of the evidence presented
<i>Software inspections</i>	Focus	The focus of inspection method
	Error Focus	Yes or No; if “Yes”, how does it relate to the current research questions
	Requirement phase	Yes or No (Did it inspect requirement documents?)
	Benefits	Benefits of the inspection method
	Limitation	Limitations of the inspection method
	Evidence	The nature of the evidence presented
<i>Human errors</i>	Human errors and classifications	Human errors and classification studies
	Errors attributable	Human errors that can be attributable to software development errors
	Related Faults	Software faults that can be caused by human errors
	Evidence	The nature of the evidence presented

## APPENDIX C DATA SYNTHESIS TABLES

This appendix reports the data extracted from selected papers and grouped according to the search focus in order to address research questions from Section 3.1. Tables 13-17 summarize the data. Description of each table and the information contained in it follows:

- Table 13 describes different quality improvement approaches that utilize error information. This search used search strings 1 and 7 (as shown in Table 3). The table consists of attributes namely: authors, name, main characteristics, error focus, limitations, relevance, and list of related references.
- Table 14 describes the various sources of requirement errors as well as other software errors (e.g., design and coding errors). This search used search strings 2 and 3 (as shown in Table 3). The table consists of attributes namely: authors, name, main characteristics, error focus, limitations, relevance, and list of related references.
- Table 15 summarizes the human error literature relevant to this research. This search used search strings 5 and 6 (as shown in Table 3). Table 15 presents the human error results characterized by: author, name of the classification, description of the error classification and the list of related references.
- Table 16 analyzes different requirement inspection techniques to determine if any of the present techniques use error information. This search used search string 4 (as shown in Table 3). The inspection techniques are summarized by: author, name, application context, description, error focus and list of related references.
- Table 17 summarizes the defect classifications that were analyzed to abstract the underlying causes of the defects (i.e., errors). This search used search string 3 (as shown in Table 3). Each classification is characterized by: author, classification name, application context, description, and list of related references.

**Table 13 - Review of Different Quality Improvement Approaches**

<b>Authors</b>	<b>Quality Improvement Approach</b>	<b>Main Characteristics</b>	<b>Error Focus</b>	<b>Limitations (if any)</b>	<b>Relevance</b>	<b>Ref.</b>
Bhandari, Halliday, Tarver, Brown, Chaar, Chillarege	Defect based Software Process Improvement	A case study of a software process improvement method that uses an attribute focusing -based analysis of defect data to help a project team make process corrections in real time	Analyzes causes of defects to suggest improvements and provide information of some of the causes	Uses defect types that addresses later stage defects rather than requirements phase defects. The attribute focusing technique is not complete, so not all problems can be found.	Some of the causes of defects were analyzed for use in the requirement error taxonomy	[19, 20]
Basili, Rombach	Goal Oriented Approach for Process Improvement	A five-step methodology for software process improvement that uses analysis of defect data to tailor the software process to the specific project goals and environments	Describes errors as problems in human thought process while trying to understand the given information, to solve problems, or to use methods and tools	This process suffers from process feedback problems, defect model corroboration problems and relies too much on historical data. In addition, it is difficult to detect problems unique to experiences on the current project	The error information provided was an input for the requirement error taxonomy	[10]
Lezak, Perry, Stoll	Root Cause Analysis	A case study analyzing the sources of defects in an effort to either prevent defects or to detect them during developing	Focus on abstracting the human-based root causes that are responsible for defects	It is a time intensive activity, requires substantial resource investment, results in a large amount of actions, is impractical for analyzing a large amount of defects.	Provided useful information about human errors that cause software defects and reported the significant influence of human errors on defect injection	[88]
Card	Defect Causal Analysis	Describes a team-based causal analysis approach for systematically improving the quality of software produced by analyzing software problem reports to prevent defects	Used a defect causal process to abstract the causes of defects, and developed some error-cause categories to identify the problem areas	Requires extensive documentation (both of the problem reports and the results), is based on sampling the problem reports and does not necessarily represent all errors, requires extensive training and investment. The resulting cause-categories are too general and incomplete	Used the information about different cause categories and other example causes for as inputs to the requirement error taxonomy	[30]

Grady	Software Failure Analysis	Analysis of defects and brainstorming the root causes of those defects, incorporating learning, and process changes. This process is more valuable than subjective assessments	Combination of failure analysis (evaluation of defect patterns) and root cause analysis is used to understand root cause(s)	Builds on the root cause analysis and suffers from the same limitations	A few of the causes made useful contributions to the requirement error taxonomy	[57]
Nakashima, Oyama, Hisada, Ishii	Software Bug Analysis	Analysis of the bugs found in the software developed by a group to find the real causes of the bug. Describes countermeasures for preventing the bugs and methods for implementing those countermeasures	Analysis of software bugs revealed many useful defect causes (errors)	Concentrates on later stage (i.e. design and coding) bugs and their causes, and the bug cause classification is not complete	Used the information about errors abstracted from bugs as an input to the requirement error taxonomy	[105]
Jacobs, Moll, Krause, Krusters, Trienkens, Brombacher	Defect Causal Analysis (using experts)	Uses accumulated expert knowledge as a substitute for an in-depth defect causal analysis on a per-object basis	Uses expert knowledge to identify the causes of defect injection	Analysis not based on project-specific defect data, requires a lot of meetings among the experts, and is developed by virtual teams	The information about defect causes was used as an input to the requirement error taxonomy	[68]
Lanubile, Shull, Basili	Error Abstraction Process	An empirical study evaluating the process of abstracting errors from faults and then using that error information to improve the quality of the requirements document	Analyzes defects to determine their causes (errors) and use that information to find additional defects	Does not provide developers with a description of various types of errors. Relies on the creativity of the developers to abstract errors from faults	Used as a starting point for the approach taken in the current research.	[86]
Mays, Jones, Holloway, Studinski	Defect Prevention Process	Improves quality by preventing the injection of defects. Includes a causal analysis to identify root cause of defects and suggest preventive actions.	Uses causal analysis to determine the specific cause(s) of a defect and provides some categorization of defect causes	Causal analysis is a cost-intensive and human-intensive process. It is practical for only a small portion of the defects, thus it is only a partial feedback mechanism.	The cause categories were used as an input to the requirement error taxonomy.	[96]
Kan, Basili, Shapiro	Total Quality Management	A management philosophy for achieving long-term success by linking quality with customer satisfaction. Key elements include achieving total customer satisfaction, continuous process improvement, considering human side of quality and driving continuous improvement in all quality parameters	It uses Mays' Defect Prevention Process for continuous improvement of the development process by injecting intelligence into the development process.	It is based on Mays' Defect Prevention Process and therefore has same problems. Also it is a long-term objective that varies in implementation.	Supports the claim that human factors are important for software quality and described some of the human factors that are important to software organization.	[73]

**Table 14 - Survey of Requirement Errors**

<b>Author(s)</b>	<b>Study Description</b>	<b>Error Type</b>	<b>Error Description</b>	<b>Ref.</b>
Lezak, Perry, Stoll	A retrospective root cause defect analysis case study that describes the defects and root cause classification	Human root causes of defects	Change coordination, lack of domain knowledge, lack of system knowledge, lack of tools knowledge, lack of process knowledge, individual mistake, introduced with other repair, communication problems, missing awareness for need of communication	[88]
Huang, Chang, and Christenen	Problems with the current requirement traceability methods and description of event based traceability is provided	Requirement traceability problems	Identification error, update error, inclusion error. Also included are problems due to lack of coordination between team members, lack of visibility into the current state of dependencies, lack of training, and the loss of links due to perceived lack of immediate benefits	[65]
Sutzke, Smidts	A stochastic model that can predict the human errors, and the description of influencing factors on the parameters of model is suggested	Influencing factors to development errors	Experience, capability, software complexity, schedule pressure, use of methods/notation, poor communication skills, team relationships, management style, process integration method, requirements volatility	[134]
Card	A team-based defect causal analysis of software problem reports. Helps prevent defects and ensure early detection	Fault cause categories	Causes of faults mainly fall into four categories as methods (which may be incomplete, ambiguous, wrong or un enforced), tools and environment (which may be clumsy, unreliable, or defective), people (who may lack adequate training and understanding) and input and requirements (which may be incomplete, ambiguous or defective)	[30]
Browne, Ramesh	Describes classes of difficulties in determining systems requirements, along with the underlying with behavior and cognitive theories	Classes of difficulties in determining requirements	Four classes of difficulties include constraints on humans as information processing (cognitive biases, satisfying, faulty reasoning, atomicity, problems in recall), Variety and complexity of requirements, Communication problems and Unwillingness of users to provide requirements (motivational biases, Hawthorne effect)	[26]
Bhandari, Chaar, Chillarege, Grady	Experience using approaches that rely on both machine and human interpretation of defect data for in-process correction and improvement	Fault causes from seven different projects	Lapses in communication between designers who were working in separate teams, lack of consensus among designers, missing item in requirement list was responsible for incomplete component-level design, lack of experience of reviewers, no knowledge of environment, new environment was not accounted for in the original system design, lack of formal requirements process, and many other causes of defects.	[19, 57]
Norman	Principles of system design are developed using analysis of one class of human errors (slips of action) that can minimize the error occurrence and effects	Classes of slips in system design	Mode errors (erroneous classification of the situation), description errors (insufficient specification of an action), lack of consistency, capture errors (overlap in the sequence required for performance of two different actions), activation errors (inappropriate actions get performed or appropriate actions failed)	[109, 110]
Endres	An analysis of defects and their causes in system programs.	Cause categories	Technological causes, organizational causes, historical causes, group dynamic causes, individual causes and other causes	[43]
Nakashima, Oyama, Hisada, Ishii, Mohri	Analysis of bugs found in software developed by a group to identify the real causes to improve quality	Causes of software bugs	Omitted judgment/ judgment missing, mishandling of decision and step, error in internal specification, human error, function specification is not updated, confirmation is missing, check is missing, assumed to be too easy, then tend not to pay much attention, preoccupied thought, assumed that they should be handled and corrected in next phase and some others	[102, 105]
Jacobs	Causes of defect injection, missed detection or late detection are	Defect Cause categories	Identified four major categories of communication causes, process causes, organization causes and technology causes with different causes	[68]

	investigated using approach that relied on accumulated expert knowledge.			
Zhang, Pham	Findings of empirical research from thirteen companies participating in software development identified 32 potential problem factors	Factors affecting software reliability	Some of the factors include development team size, work load, work standards, development management, environment, relationship of detailed design and requirement, volume of documentation, skills, resource allocation, methodologies, tools, requirements analysis, input/output device, storage device, human nature (mistake and omission) and many other factors along with their correlated factors and their rankings are listed	[149]
Oliveira, Zlot, Rocha, Travassos	Emphasizes the importance of understanding of domain of a system and defines the concept of “Domain oriented software development”	Domain Errors	Lack of application domain knowledge by the software team, high turn over of the software teams, lack of knowledge about typical activities performed in the domain, and misunderstandings due to working with several different software systems and domains	[111]
Beecham, Hall, Britton, Cottee, Rainer	Presents a software process improvement model that represents key requirement engineering practices. Validated by experts to better understand the requirements process	Requirement engineering problem classification	Requirements engineering problems are categorized into organizational problems (i.e. culture, developer communication, resources, skills, staff retention, training and user communication) and technical problems (i.e. complexity of application, poor user understanding, requirement growth, requirements traceability, undefined requirements process, vague requirements).	[14]
Ko, Myers	Proposed a framework and methodology focused specifically on errors to support the description and identification of causes of software errors in terms of chains of cognitive breakdown	Software errors	Inattention (strong habit intrusion, interruptions, delayed action, exceptional stimuli, interleaving), over attention (omission, repetition), wrong rule (problematic signs, information overload, favored rules, favored signs, rigidity), bad rule (exception proves rule, wrong action), bounded rationality (selectivity, biased reviewing, availability), faulty models of problem space, task complexity, strategic variability, level of detail in task specification	[79]
Debou, Combelles	Presents problems in requirements management and project management gathered from experiences and lessons learned from industry sources	Management problems	Requirements management (change requests insufficiently formalized, impact analysis not systematically achieved and poorly documented, decision to implement changes not necessarily taken at right level, absence of process measurement), Project management ( lack of precision, no historic data, poor planning, improper risk evaluation, late actions)	[37]
Robinson, Pawlowski	Problems of requirement inconsistencies are provided along with the techniques that can manage requirements document inconsistency	Requirement inconsistency problems	Requirement inconsistencies can be due to the technical problems (voluminous/ changing requirements and analyst, complex requirements) or social problems (conflicting/ changing/ unidentified stakeholders, changing expectation) of constructing a requirements document	[120]
Hall, Beecham, Rainer	Study of problems experienced by twelve software companies in the requirement process is discussed to better understand requirements process	Requirement problems	Requirement problems are classified as organizational based (communication, inadequate skills and resources, staff retention, user communication, lack of training, company culture) and requirement process based problems (vague initial requirements, undefined requirements process, requirements growth, complexity of application, poor user understanding)	[59]
Smith	An excerpt from the author’s book, “Troubled IT projects, prevention and turnaround”, describes 40 troubled project root causes	Root Causes of troubled IT projects	Root causes at different stages of software process (project conception, project initiation/mobilization, system design, system development, system implementation, and system operation-benefit delivery-disposal) are listed and can not be contained in this table	[132]
Basili and Perricone	Distribution and relationships derived from data collected during development	Software Errors	Misunderstanding of requirements, mistaken assumptions about value or structure of the data, misunderstanding of external environment, clerical error, error due to previous miscorrection	[8]

	produces some surprising insights into the influencing factors in software development		of error, mistake in control logic or computation of an expression	
Endsley	Taxonomy of situation awareness errors at 3 different levels is presented that describes many factors that can lead to these errors	Decision making errors	Inability to correctly perceive information (data not available, data hard to discriminate, misperception of data, memory loss), inability to correctly integrate or comprehend information (poor mental model, use of incorrect mental model, over-reliance on default values, other), inability to project future actions, not able to maintain multiple goals	[44]
Sasou, Reason	Considering large complex systems, a taxonomy of team errors and their relationship with performance shaping factors (PSF) is provided	Team errors	Team errors can be individual errors (independent and dependent) or shared errors (dependent and independent). Different kinds of errors in all the 4 classes are presented in detail and can not be contained in this table	[124]
Viller, Bowers, Rodden	Survey of contribution of human errors from social and organizational literature is done to improve requirement engineering process and development process for dependable systems	Requirement engineering errors	Error in individual work (recognition/ attention/ memory/ selection slips, mistakes, violations), Group performance (leadership errors, normative & informational influence, minority influence), Organizational (incompatible goals, inadequate deficiencies, inadequate communications, poor planning and scheduling, poor operating procedures, poor training)	[144]
Basili, Rombach, Katz, Yap	Classifies errors by project aspects that caused problems (is further refined in another paper)	Errors	Application errors (misunderstanding of application or particular domain), Problem-solution errors (not knowing, misunderstanding, or misuse of problem solution processes), semantic errors, syntax errors, environment errors, information management errors, clerical errors	[9, 10]
Gaitros	Presents some of the major causes of failure in large software development projects and their preventive measures	Common errors	Not knowing what is needed, unclear lines of communication and authority, political reasons, weak management structure, lack of experienced and capable management, improper work environment, inexperienced technical staff, slow start, not involving users at all stages	[53]
Mays, Jones, Holloway, Studinski	Defect prevention process uses causal analysis to identify root causes and suggests preventive actions and is implemented within IBM	Error cause categories	Oversight (not able to consider all cases and conditions), Education (misunderstanding of some aspect of process or product), Communication , Transcription ( developer knew what to do but simply made the mistake) and flawed process errors	[96, 146]
Lutz	Analyzes the root causes of safety-related software errors in safety-critical, embedded systems to help reduce these errors in future	Root causes	Communication errors (within a development team or among teams), misunderstanding of interfaces and assumptions, errors in recognizing and deploying requirements, inadequacies in communication or development methods, complexity errors, mistakes in application	[91]
Lutz	Provides a safety checklist for analysis of software requirements and analyzes the common causes	Safety related software errors	Misunderstandings of interface requirements, lack of requirements for robustness, implementation / operator errors, misunderstanding of dependencies, use of obsolete data for control decisions and handling of overload / saturation	[92]
Jaffe, Leveson, Heimdahl, Melhart	Defines a set of criteria to help find errors in software requirements specifications with focus on properties of robustness and ambiguity	Software requirement errors	Errors due to assumptions about the behavior of the parts of system, assumption (environmental and timing) errors, safety errors (inadequacy and lack of design foresight), control errors, documentation errors, inability to distinguish between observably distinct behavior patterns, and not able to cover all possible circumstances	[69]

**Table 15 - Survey of Human Error Literature**

Author	Taxonomy	Description	Ref.
Reason	Classification of slips, lapses, mistakes and violations.	Classification consists of Mistakes, Slips and Lapses. <i>Mistakes</i> are due to planning problems i.e., wrong plan is used. <i>Lapses</i> occur during the storage of information i.e., intention is not arrived or recalled on time or at all. Finally, <i>slips</i> occur during execution of action i.e., the plan is correct but the execution is wrong because the action is not appropriate to the intention.	[117]
Rasmussen	Skill, Rule and Knowledge based taxonomy.	Skill category describes errors that use a predefined solution or skill to achieved desired solution. Errors at skill level concern attention and memory. Errors at rule-based level can be caused by memory problems, often not recognizing appropriately the context for applying a rule, and inadequate formation of rules. Errors at knowledge-based level occur due to inadequate knowledge. Errors at this level are due to individual human problem solvers.	[117]
Senders and Moray	Three different taxonomies: Phenomlogical taxonomies, Cognitive taxonomies, Deep Rooted Tendency taxonomies that describe the how, what and why concerns of errors. Proposed four classification levels to address these taxonomies.	Phenomenological taxonomies classify errors in terms of observed events e.g. omissions, substitutions, unnecessary repetitions etc. Cognitive taxonomies describe the errors according to the stages of human information processing e.g., perception, memory, attention. Deep-rooted tendency taxonomies describe errors based on the biases. In addition, the four levels proposed includes phenomenological level, hypothetical internal processes level, neuron-psychological mechanisms level and the external processes level.	[127]
Reason	Used the Rasmussen model to develop a generic error modeling system (GEMS) of human errors	In GEMS model, the cognitive control process operates through a cognitive continuum. The structure is a model of unsafe acts. There are 2 unsafe acts namely intentional and unintentional. Further, unintentional acts include slips and lapses whereas intentional acts include mistakes and violations. Further different slips, lapses, mistakes, and violations are described. Reason includes violation in addition to Rasmussen categories and distinguishes between slips and lapses at skill base level.	[117]
Swain and Guttman	Discrete action taxonomy, a simple classification for individual discrete actions.	Authors developed a simple taxonomy describing the errors that individual commits if they omit something, perform a wrong selection of control, perform actions out of sequence, or perform actions either too early or too late. The error categories include omissions, commissions, sequence errors, and timing errors.	[136]
Hollnagel	Phenotypes and Genotypes	Scheme describes the manifestations of a cause (which he called phenotypes) and the underlying causes (which he called genotypes). Provides a mapping from the actual errors to its manifestations.	[64]
Fitts and Jones	Control errors taxonomy	Six categories include substitution errors, adjustment errors, forgetting errors, reversal errors, and unintentional activation errors, unable to reach control errors.	[47]
Gabrowski and Roberts	Determinants in error adverse systems	Presents set of determinants of error-adverse systems namely structure, decision making, communication, human-computer interfaces and culture. Identifies and describes general areas for human and organizational error identification for large systems.	[56]
Cacciabue	A taxonomy of erroneous behavior	Taxonomy of erroneous behavior is described in the terms of system-related causes, and personal-related causes. Further error modes and causes are described in relation to the execution function, planning function, and interpretation function and observation function. The correlation between cause and effects of erroneous behavior is described.	[29]

Galliers, Minocha, Sutcliffe	Taxonomy of influencing factors / preconditions for occurrence of error.	The influencing factors are grouped into 4 main categories as environmental conditions, management & organizational factors, task/domain factors, and user/personnel qualities. Different problems for slip and mistake type errors are described within each category.	[54, 55]
Sutcliffe and Rugg	Taxonomy of error types that builds on work by Reason and Hollnagel.	Authors describe their error taxonomy using different levels of error categories. Error categories includes operational description, cognitive causal categories, social and organizational causes, and design errors. Within each error category, different kinds of errors are described and similar errors are grouped together. Also authors describe the error analysis of all the errors in each category of error classification taxonomy.	[135]
Norman	Categorization of action slips	Classified slips into 3 categories namely formation of intention, activation and triggering. In addition, there are 2 kinds of errors within formation of intention category i.e. errors in classifying the situation and errors that result from ambiguous or incompletely specified intentions. Also, there are 2 kinds of slips from faulty activation of schemas namely unintentional activation and loss of activation. Finally, the errors due to fault triggering includes schema triggered at wrong time, spoonerisms, confusing intention with action.	[109, 110]
Norman	Stages of Interaction cycle	Classified problems in terms of phase of the interaction cycle. The different categories include errors during “intention”, “action”, “execution”, “perception”, “interpretation,” and “evaluation”.	[108, 112]
Shorrock and Kirwan	Human Error Identification tool, cognitive errors, and inter-related error taxonomies.	Task error taxonomy comprising of 13 error categories, along with the information taxonomy and performance shaping factors taxonomy. Then it describes external error mode taxonomy. All these taxonomies are described based on a cognitive framework with cognitive domains namely perception, memory, judgment-planning-decision making, and action execution. All the different human errors are described within these four cognitive domains using different human error taxonomies.	[130]
Issac and Bove	HERA and detailed taxonomies of error.	HERA (Human Error Reduction) is a technique that analyses and describes the human errors in air traffic management domain. HERA contains much elaborated taxonomies for error analysis and has been adapted to ATM system. It consists of 5 cognitive domains namely “perception & vigilance”, “working memory”, “long-term memory”, “judgment, planning and decision-making”, and “response execution”.	[25, 67]
Stanton and Stevenage	Human error identification (HEI) technique literature and SHERPA (HEI tool)	This paper provides literature regarding human error identification and describes a particular tool for human error identification called SHERPA. SHERPA utilizes an error classification taxonomy that has errors classified into the categories of “action”, “checking”, “retrieval”, “communication” and “selection errors”.	[133]

**Table 16 - Survey of Software Inspection Techniques**

<b>Inspection Technique</b>	<b>Application Focus</b>	<b>Description</b>	<b>Error Focus</b>	<b>Ref.</b>
Fagan Inspection	Design and Code Inspections	Lays the foundation for the development of various inspection techniques. Describes a basic inspection process for reducing errors in design and code inspection. It talks about the inspection process in detail (inspections at design and code level)	<b>NO</b>	[3, 45]
Phased Inspection Process	All documents.	It came as an improved inspection process in which the inspection checklist is divided into series of smaller, focused phases (steps) in a definite order. The aim was to provide rigorous, repeatable, process with results that can be guaranteed.	<b>NO</b>	[3, 78]
Ad-Hoc technique	Requirement Inspection.	Ad-hoc technique provides no specific guidance to the reviewers on how to read the document and what to look for? It relies more on the skill of individual reviewers on locating defects	<b>NO</b>	[3]
Checklist Based Reading technique	Requirement Specification.	Contrary to an Ad-hoc technique, Checklist-Based Reading provides inspectors with a checklist of questions to assist the reviewer in detecting defects while reading the software document.	<b>NO</b>	[2, 3, 17, 32, 52, 83-85, 114]
Defect Based Reading technique	Requirement Specification.	The Defect Based Reading technique provides reviewers with different fault classes. Rather than providing a checklist of questions, it provides list of fault classes. Reviewers inspect the documents looking for defects in these classes. It is applied successfully to requirement specifications.	<b>NO</b>	[3, 17, 52, 85, 114, 140]
Usage Based Reading technique	Requirement Specifications.	The Usage Based Reading technique uses the use cases to help reviewer inspect the requirement document from the user's point of view. It helps the reviewer focus more deeply and detect faults that are more critical.	<b>NO</b>	[3, 17, 114, 139, 140]
Function point Reading technique	Requirement Specifications.	The Function point reading technique is another variant of scenario based reading techniques. It involves constructing domain related questions for all key issues, further supplemented by checklist and it is used to develop scenarios to investigate different aspects of SRS.	<b>NO</b>	[3, 31, 84]
Reading by Stepwise Abstraction	Code Documents	Code reading by stepwise abstraction identifies prime subprograms in the software, determines their function, and uses them to determine a function for the entire program. Then it is possible to compare this derived function to the intended function to find defects.	<b>NO</b>	[3, 12, 84, 89]
Traceability Based Reading Technique	Requirements, Design documents	Traceability Based Reading is used to verify consistency between the design and the requirement documents (vertical reading), and to ensure internal correctness of the design (horizontal reading)	<b>NO</b>	[140]
Perspective Based Reading technique	Requirement, design documents, functional code modules, OO artifacts	Perspective Based Reading is a scenario based reading technique that provides procedural guidance, tailored to natural language requirements. The whole process entails selecting a set of perspectives, tailoring procedures for each perspective to build an abstraction of the requirements. Then, it describes questions for finding defects for each model and reviewers use it to review document	<b>NO</b>	[3, 12, 17, 52, 82-84, 114, 131, 140]
Metric-Based Reading technique	Requirement Inspection	The Metric based reading technique detects specific kinds of defects in natural language software artifacts using a set of rules or heuristics that describe the relationship between metric values and presence of certain types of defects. Inspectors use these heuristics or set of rules to detect defects. Experiments show that this approach helps in detecting more defects than checklist review	<b>NO</b>	[18]

Task Based Inspection technique	Code inspection	The Task Based Inspection technique inspects complex computational code, but can also be used in other domains. It focuses on work of individuals rather than the team and is based on the observation that providing structure and guidance to individual inspector can improve effectiveness.	<b>NO</b>	[75]
Inspection using Error Abstraction	Requirements Documents	This technique involves detecting faults using a taxonomy of faults followed by abstracting the causes of faults detected in first step and then using the error information to re-inspect the document and locate more faults that were over looked first time. It rests heavily on the creativity and reviewers ability to abstract causes of faults.	<b>YES</b> , we are building over this work by developing error classification.	[86]
N-Fold Inspection Process	User Requirements Document	The N-Fold inspection employs formal inspection process replicated in parallel using N independent teams. It is based on the hypothesis that there is little overlap between faults found by each parallel team and each team detects different faults.	<b>NO</b>	[3, 94, 126]

**Table 17 - Survey of Defect Taxonomies and Defect Classes**

Author	Classification	Application Context	Description	Ref.
Chillarege, Bhandari, Chaar, Halliday, Moebus, Ray and Wong.	Orthogonal Defect Classification (ODC)	Addresses mainly code defects; and is difficult to use for requirements and high-level design.	Orthogonal Defect Classification is a concept that enables in-process feedback to developers by extracting signatures on the development process from defects. This paper described classification of defects along with the necessary and sufficient conditions required to provide feedback to a developer. The classification has attributes namely defect types, missing/incorrect, trigger, source, and impact	[19, 33]
Grady	Hewlett-Packard defect classification scheme (HP)	All Stages.	HP classification is described in a set of three-layered architecture namely “origin”, “type” and “mode”. The first step is to determine the activity that introduced the defect to assign origin attribute, then the type attribute describes the defect in detail, in the last step, attribute mode describes the kind of defect and why it was a defect.	[50]
Schneider	Fault Classification	Requirement Documents	Classified user requirement faults into 2 classes. <i>Missing Information</i> includes faults of missing functionality/feature, missing interface, missing performance, missing environment. <i>Wrong Information</i> includes faults of ambiguous information and inconsistent information.	[17, 126]
Basili	Fault Classification	Requirements Document	Classifies faults into 8 fault classes: Clerical, Ambiguity, Omission, Inconsistency, Incorrect Fact, Information put in wrong section, Implementation fact included and Other.	[7, 17]
Bell and Thayer	Requirement Fault categories	Requirements Document	Uses information from problem reports to categorize requirement problems into 8 distinct categories: Requirement out of scope, Missing/incomplete/inadequate, Requirement incorrect, Inconsistent/incompatible, New/changed requirement, Requirement unclear and typographical.	[15]
Sakthivel	Requirement fault classification	Requirements Document	Describes a detailed requirement faults taxonomy building on other fault classifications. It has 6 main classes: Incomplete, Inconsistent, Infeasible, Untestable, Redundant and Incorrect. Within each fault class, different kinds of faults are described along with an example.	[122]
Ackerman	Requirement Faults	Requirements Document.	A requirement checklist that categorize requirement faults into 3 categories: Completeness, Ambiguity and Consistency. Under each question are different questions to help locate each kind of faults.	[2]
Porter	Requirement faults	Requirements Document	For ad-hoc and checklist, authors use fault classes of omission & commission. For scenario inspection, requirement fault are Data type inconsistencies, Incorrect functionality and Missing or Ambiguous functionality.	[114]
Hayes	NASA’s requirement fault taxonomy	Requirements Document	Classifies software faults in 13 categories: Incompleteness, Omitted/Missing, Incorrect, Ambiguous, Infeasible, Inconsistent, Over-Specification, Not Traceable, Unachievable Item, Non-Verifiable, Misplaced, Intentional Deviation and Redundant or Duplicate.	[61]