Journal of Computer Science

Review Paper

Defect Types and Software Inspection Techniques: A Systematic Mapping Study

[1]Ricardo Theis Geraldi and [2]Edson OliveiraJr

[1]Department of Informatics, State University of Maringá, Avenida Colombo, 5790, Maringá-PR, Brazil
[2]Department of Informatics, State University of Maringá, Avenida Colombo, 5790, Maringá-PR, Brazil

Article history
Received: 31-07-2017
Revised: 08-09-2017
Accepted: 09-10-2017

Corresponding Author:
Edson OliveiraJr
Department of Informatics,
State University of Maringá,
Avenida Colombo, 5790,
Maringá-PR, Brazil
Phone: +55 (44) 3011-5121
E-mail: edson@din.uem.br

Abstract: Software inspection has been used to guarantee and control the quality of products by detecting defects, which can be spread out throughout the entire software life cycle. Therefore, the main premise is to identify and reduce the number of defect types in software artifacts during inspections. This work focuses on providing an up to dated overview of existing defects in the context of software inspection techniques. A systematic mapping was carried out, from which 2096 primary studies were retrieved and 32 were final selected. From the analysis, classification and aggregation of the retrieved studies, important different defect types were identified. Most studies encompass defect types by means of experiments and proposed techniques and approaches. Thus, as a main result, the identification of several different studies with distinct proposals concerned on defect types is evident. Although researchers have conducted studies over time, a general pattern on the detection of defects could not be established. Therefore, the scenario in which this study was carried out provides researchers with the capability of conducting further research in a motivating and challenging research topic, as well as practitioners with the adoption of empirically evaluated inspection techniques and respective defect types.

Keywords: Defect Type, Software Inspection, Systematic Mapping Study

Introduction

A software system can incorporate different defect types that must be detected and removed throughout its life cycle. Such defects increase development and maintenance costs (Boehm and Basili, 2001). In this context, verification and validation activities and dynamic and static analysis are essential as they contribute to software quality control (Sommerville, 2015). Effective software inspections, thus, increase such a control.

In this scenario, software inspection is a singular type of software review conducted by inspectors and formally applied in different software artifacts to maximize the number of defects found, in order to minimize defects in software systems to be delivered (Cheng and Jeffery, 1996; Fagan, 1986).

By means of software inspections, it is possible to verify several elements from software artifacts in order to detect different defect types existing in software specifications (Gilb and Graham, 1993; IEEE, 1998b). It is essential that developers become aware of inspections to iteratively improve the quality of a software product (Anda and Sjøberg, 2002; Souza *et al*., 2013).

Defect types are necessary to identify persistent fault in different software artifacts. Thus, main pre-defined defect types are useful to inspection techniques, for guiding the detection of possible defects. The objective of main predefined defect types is to specify and ensure which or at least the major defects known in the literature are detected at the time of inspections (Alshazly *et al*., 2014).

Main existing defect types in the literature can be highlighted (Fagan, 2002): *Omissions*, *Incorrect facts*, *Inconsistencies*, *Ambiguities and Extraneous information*. Thus, detecting these defect types might provide evidence for refining inspection techniques in an adaptation perspective.

Based on the aim of software inspections and defect types, it seems essential to map the literature related to this research topic in order to provide a knowledge body for researchers and practitioners improving and/or incorporating inspection activities in emerging and state-of-the-art techniques, such as software product lines and model-driven engineering. Therefore, this paper presents a Systematic Mapping (SM) study, which aims at identifying existing defect types and inspection techniques

Science Publications

© 2017 Ricardo Theis Geraldi and Edson OliveiraJr. This open access article is distributed under a Creative Commons Attribution (CC-BY) 3.0 license.

and studies that provide evidence of such software inspection techniques and respective defect types.

The results of this systematic mapping identified what defect types are most frequent in software inspection techniques based on different domains. In addition, results evidenced the identification of several different studies with distinct proposals concerned on defect types, as well as many empirical studies.

This paper is structured as follows: Study background is presented in Section 2; the systematic mapping process is detailed in Section 3; obtained results are discussed in Section 4; threats to validity for this study are discussed in Section 5; and concluding remarks are presented in Section 6.

Background of the Study

Software inspection is a specific type of software review (Ciolkowski *et al*., 2003) applied to artifacts by means of a systematic and well-planned defect identification process (Fagan, 1986; Kalinowski and Travassos, 2004; Sauer *et al*., 2000). According to CeBASE (http://www.cebase.org/defect-reduction.html), over 60% of defects can be identified at early stages of the software life-cycle (Boehm and Basili, 2001; Gilb and Graham, 1993).

Fagan (1986) proposed a software inspection process driven by roles, such as moderator, inspector and author and activities. Once the process is established, one or more review techniques can be taken into account for performing inspection activities (Pressman, 2014; Sommerville, 2015).

The process of defect detection must be standardized and non-ambiguous for the artifact under revision, for instance, the requirements specification (IEEE, 1998a). The study of van Lamsweerde (2009) is evident in the literature by the singularity of the established classification scheme of defect types. This classification can be used independently of the inspection techniques, usually adapted for detecting defects based on requirements engineering.

The taxonomy of defect types is taken into account by several more studies, such as in Alshazly *et al*. (2014; Silva and Vieira, 2016; Teixeira *et al*., 2015; Travassos *et al* ., 1999; Walia and Carver, 2009). For example, the IEEE Standard 1012-2012 recommends verifying and validating an inspected system throughout defect types, such as: Complete, Correct, Omissions, Ambiguities, Traceable, Testable and Consistent (IEEE, 2012).

Furthermore, Hayes *et al*. (2006) proposed a taxonomy to detect defects based on the study of Miller *et al*. (1998) that can be adapted for software inspection techniques with regard to the following defect types: Requirements, Incompleteness, Omitted/Missing, Incorrect, Ambiguous, Infeasible, Inconsistent, Over-specification, Not Traceable, Non-Verifiable, Misplaced, Intentional Deviation and Redundant.

For example, to explain the most common defect types in the literature, the study of Anda and Sjøberg (2002) proposes a taxonomy for the following defect types for checklist-based software inspections mostly applied to functional requirements and use cases diagrams:

- *Omissions*, absence of a mandatory element or functionality, e.g., variations of a certain requirement not present in the specification
- *Incorrect Facts*, a functional requirement or use cases incorrectly described
- *Inconsistencies*, problems from functional requirements or use cases with their goals and specifications poorly designed - e.g., descriptions, variations and terminology
- *Ambiguities*, a specified functional requirement or use case does not meet its objective - e.g., descriptions with multiple interpretations or misdescribed
- *Extraneous Information*, functional requirements or use cases are redundant - e.g., are duplicated and without specification
- *Consequences or Others*, unexpected problems in the specification of functional requirements or use cases - e.g., communication between analysts and developers is flawed with respect to the project objectives and what should or not be done

Travassos *et al*. (1999) applied defect types to their proposed technique, named Traceability-Based Reading (TBR), taking the proven effective set of defect types from Anda and Sjøberg (2002) into account. In addition, Zhu (2016) discussed the impact of the defect types based on Travassos *et al*. (2001) taxonomy in high-level object-oriented designs using UML diagrams.

The secondary study conducted by Hernandes *et al*. (2013) contributes to a significant mapping of empirical studies in the area of software inspection. Thus, the identification of the main defect types in combination with mapped empirical studies is essential to provide evidence for adopting and/or proposing new inspection techniques. However, such a work is restricted to mapping only empirical studies and controlled experiments, whereas this mapping study is concerned on the overall mapping of defect types and inspection techniques.

Based on the results, this systematic mapping provides an identification of studies focused on defect types. Thus, these defect types can be adapted or applied to different techniques proposed in the software inspection scenario. Therefore, inspections may be guided by means of a well-organized set of defect types to improve the quality of inspections.

Systematic Mapping Process

The Systematic Mapping (SM) carried out in this paper is aimed at providing researchers and practitioners with an overview of primary studies of defect types in software inspection activities (Kitchenham *et al*., 2010; Petersen *et al*., 2008a).

Petersen *et al*. (2008a), by comparing different methods when performing systematic mappings and reviews, established six stages as a strategy for the elaboration and conduction of SMs: (I) Definition of protocol; (ii) definition of research questions; (iii) conducting the search for primary studies; (iv) screening papers based on inclusion/exclusion criteria; (v) classifying the papers; and (vi) data extraction and aggregation.

The conducted stages of this SM is in accordance to Fig. 1. Thus, this section describes how each stage was planned and conducted, which are: Research questions (Section 3.1), research process (Section 3.2), digital databases, keywords and search strings (Section 3.3), inclusion and exclusion criteria (Section 3.4), classification scheme (Section 3.5) and data extraction and aggregation (Section 3.6).

Definition of Research Questions

The main objectives of this SM were established aiming at: (I) identifying in the literature defect types used in environments/domains and software inspection techniques; (ii) presenting an overview of defect types empirically evidenced; and (iii) discussing primary studies on inspection techniques with defect types. Therefore, the following research questions are stated:

- *Research Question* (*RQ*1). What defect types have been taken into account by software inspection techniques?
    - *RQ*1*a*. Which environments or domains defect types were applied to?
    - *RQ*1*b*. Which defect types were empirically evidenced?
- *Research Question* (*RQ*2). What kind of evidence do inspection techniques/approaches that adopt classified defect types provide?
    - *RQ*2*a*. Which inspection techniques adopted classified defect types?
    - *RQ*2*b*. Which evaluation results are documented?

The Search Process

The followed search process is based on criteria and guidelines proposed by Kitchenham (2007; Kitchenham *et al*., 2010; Petersen *et al*., 2008a) with relation to performing SMs. In addition, knowledge body from several studies in different areas, such as Barney *et al*. (2012; Neto *et al*., 2011; Mohabbati *et al*., 2013; Novais *et al*., 2013), served as a basis to provide directions on how to conduct this SM.

Therefore, the search process procedures (Fig. 2) were established, as follows:

1. Selection of digital databases and indexed search mechanisms (Section 3.3)
2. Definition of keywords to compose the main query and the search strings applied to digital databases and mechanisms to retrieve primary studies. The following keywords were used: “software inspection” and “defect type” (Section 3.3)
3. Composition of such keywords and their variations using ”AND” and ”OR” operators (Section 3.3)
4. Application of defined search strings to digital databases and such mechanisms. A list with a considerable number of primary studies was retrieved. Inclusion and exclusion criteria were applied to filter such studies. Figure 3 and Table 1 Illustrates such filters in detail, as well as the number of primary studies selected based on each filter activity
5. Duplicated studies and potential conflicts, for instance, were reviewed to generate an updated list (Section 3.4)
6. Definition of a classification scheme based on categories (Section 3.5)
7. Extraction and aggregation of data (Section 3.6) by means of visualization techniques (graphs, bubble plots, etc) in order to present the obtained results. Thus, a brief discussion on the subjects related to this SM (Section 4) was carried out

A search was performed on the digital databases using the search query, respective keywords (Filter #1) and obtained 2096 studies, as presented in Section 3.3. Then, a preliminary selection was carried out by reading title, abstract, introduction and conclusion of the retrieved studies, as well as the application of inclusion and exclusion criteria (Filter #2) and obtained 86 studies. Therefore, 32 selected primary studies were fully read based on definition of research questions (RQ1 and RQ2 and derivatives), according to Filter #3 (Fig. 3).

The retrieved studies were filtered based on the inclusion and exclusion criteria according to Filter #2 and Filter #3 in Section 3.4 and Fig. 3. The first author of this paper performed reading, interpretation and selection of studies. The second author helped to interpret and decide including or excluding certain studies in case of indecision of the first author. Thus, such studies were classified (Section 3.5), as well as provided a constructive analysis (Section 4.3).

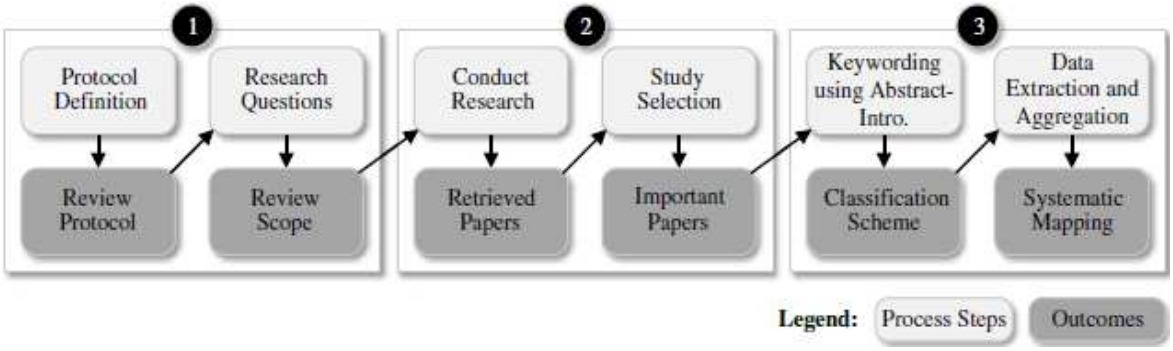

Fig. 1. The followed systematic mapping process (adapted from Petersen *et al*. (2008a))

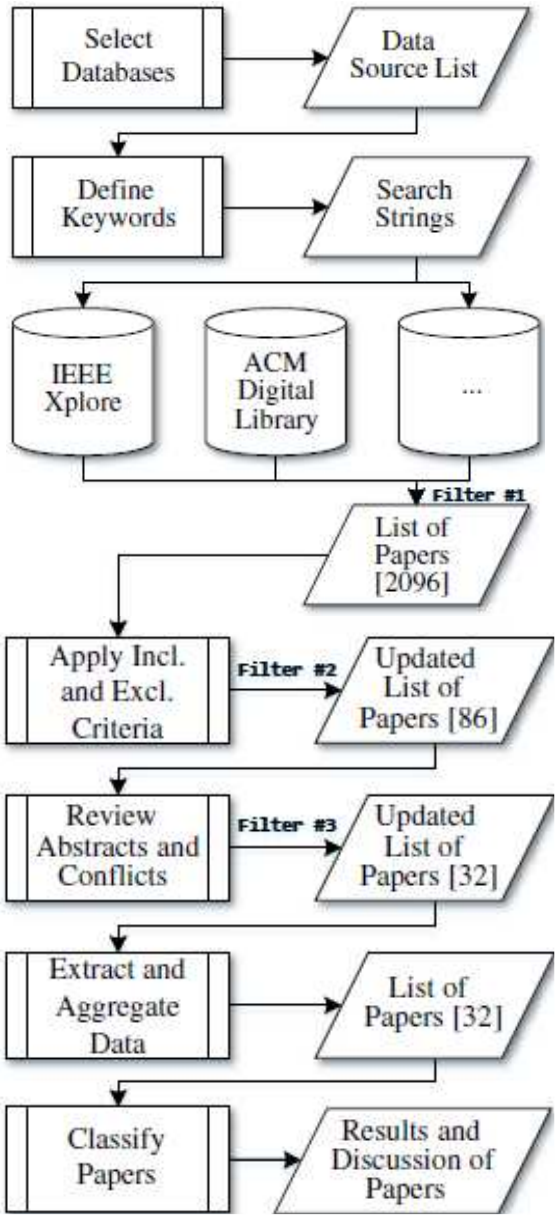

Fig. 2. The Search Process (adapted from Barney *et al*. (2012))

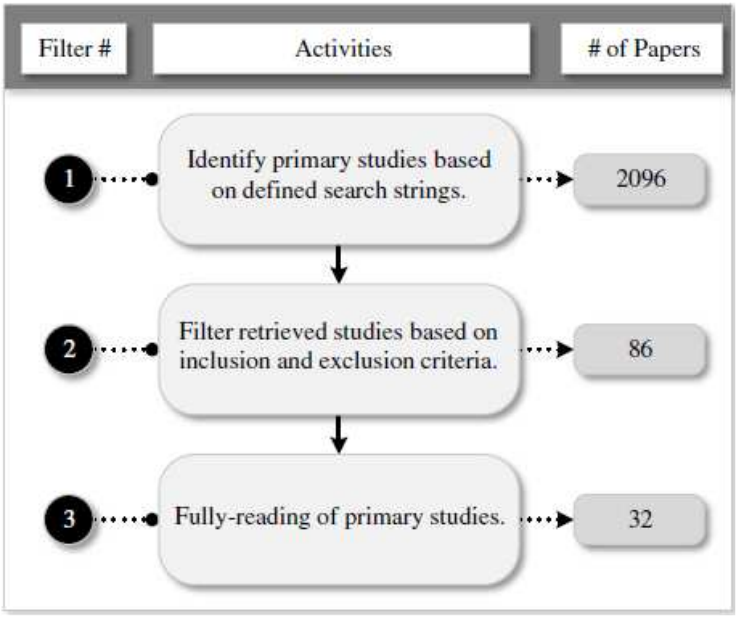

Fig. 3. Stages of Search for Primary Studies (adapted from Neto *et al*. (2011; Mohabbati *et al*., 2013))

Table 1. Number of studies per data source, filter and duplicated studies

Data source	Filter #1	Filter #2	Filter #3
ACM Digital Library	408	23	7
Compendex	197	12	5
ELSEVIER ScienceDirect	467	15	4
Google Scholar	105	7	3
IEEE Xplore	525	27	11
Scopus	827	2	2
#Duplicated Studies	**433**	-	-
Total (with duplicated studies)	**2529**	-	-
Total (no duplicated studies)	**2096**	**86**	**32**

Definition of Digital Databases, Keywords and Search Strings

Once the research questions were defined, the next step was the definition of the digital databases and mechanisms aiming at allowing search and identification of primary studies. Therefore, the set of selected digital databases is as follows: (I) ACM Digital Library- "papers published by ACM and bibliographic citations from major publishers in computing"; (ii) Compendex- "is the broadest and most complete engineering literature database available in the world."; (iii) ELSEVIER ScienceDirect- "peerreviewed full-text scientific, technical and medical content."; (iv) Google Scholar - "wide search mechanism for articles, dissertations and thesis."; (v) IEEE Xplore- "resource to discover and access scientific and technical content published by the Institute of Electrical and Electronics Engineers (IEEE) and its publishing partners."; and (vi) Scopus - "the largest abstract and citation database of peer-reviewed literature."

Next step was the definition of appropriate keywords to build search strings in order to establish a general search query. This query can be adapted for each search, in an iterative basis and applied to the selected digital databases. Thus, keywords were applied by means of the search query and they were modified according to each digital database and mechanism search tool.

Table 2 presents the search strings list and respective keywords.

Definition of Inclusion and Exclusion Criteria

Aiming at selecting the studies to contribute to answer the research questions of this paper, the following inclusion and exclusion criteria were defined:

Inclusion Criteria

For each research question, inclusion criteria were defined, as follows:

- *RQ1*. Studies that present defect types related to software inspection techniques:
  - *RQ1a*. Studies that present environments or domains with defect types applied; and

  - *RQ1b*. Studies that present defect types empirically evidenced
- *RQ2*. Studies that propose inspection techniques or approaches associated to defect types:
  - *RQ2a*. Studies that propose inspection techniques with classified defect types
  - *RQ2b*. Studies that document evaluation results

Exclusion Criteria

For each research question, exclusion criteria were defined, as follows:

- *RQ1*. Studies that do not present defect types related to software inspection techniques:
  - *RQ1a*. Studies that do not present environments or domains with defect types applied and
  - *RQ1b*. Studies that do not present defect types empirically evidenced
- *RQ2*. Studies that propose neither inspection techniques nor approaches associated with defect types:
  - *RQ2a*. Studies that propose neither inspection techniques nor used classified defect types; and
  - *RQ2b*. Studies that do not document evaluation results

In addition, the following exclusion criteria were defined: (I) Studies in languages other than English; (ii) studies, which are not in one of the following file formats: PDF, DOC or ODT; (iii) opinion and/or philosophical papers; (iv) duplicated studies, i.e., studies retrieved from more than one of the defined data sources; (v) unavailable studies, for instance, an unavailable URL; and (vi) studies with less than four pages.

Classification Scheme

In order to classify the retrieved research types in this SM, the Wieringa *et al*. (2005) classification method was adopted. Such a method is suggested by Petersen *et al*. (2008a) as it has a well-defined classification structure. The Wieringa *et al*., (2005) method performs classification of research types by means of six categories: (i) *Validation Research*: Evaluates techniques usually performed in an academic environment. Methods used to evaluate research are: Experiments, simulations, prototype constructions and mathematical analysis; (ii) *Evaluation Research*: Evaluates techniques usually performed in industry, focusing on a research, a research problem or practical technique implementation; (iii) *Solution Proposal*: Focuses on new techniques proposed and/or revised based on the research problem; (iv) *Philosophical Papers*: Aims to present new concepts that can be explored for research; (v) *Opinion Papers*: Presents positive or negative opinions of an author concerning, for instance, certain techniques, experiments, case studies; and (vi) *Experience Papers*: Presents experiences of an author with respect to particular research verified in practice.

The classification of the retrieved studies was made by reading abstract, introduction and conclusion of each retrieved study, as well as excerpts in order to make sure of the proper category of a study. Figure 4 illustrates this classification scheme.

Data Extraction and Aggregation

Data extraction summarizes data with regard to the final set of selected primary studies (Bailey *et al*., 2007). Therefore, the following metadata of each study was extracted: (I) Source: ACM Digital Library, IEEE Xplore and ELSEVIER ScienceDirect, as well as the electronic search mechanisms: Compendex, Google Scholar and Scopus; (ii) Title; (iii) Authors and afilliations; (iv) Publication year: Studies were identified until May/2017; (v) Publication type: Conference, Journal, Workshop, Book Chapter, Book, Master Dissertations, Ph.D. thesis; (vi) Publication venue acronym. In addition, we extracted the following data: (I) Defect Types (e.g., Omissions, Inconsistencies and others); (ii) Inspection Techniques (e.g., Checklist-Based Reading, Perspective-Based Reading, amongst others); and (iii) Software Artifacts (e.g., UML diagrams).

The Mendeley tool (desktop version v1.11) (Mendeley, 2014) was adopted for providing a better organization of bibliographic references (A study package for replicating this study, as well as bibtex files, selected studies URL per source and query strings per source are available at: http://www.din.uem.br/~edson/defect_types) of this study.

Table 2. Keywords and search strings composing the general search query

Search Query (SQ)
("software") AND (("inspection" AND ("technique" OR "activity" OR "strategy")) OR ("defect type" OR "type of defects" OR "defect detection" OR "requirements defect" OR "fault detection"))

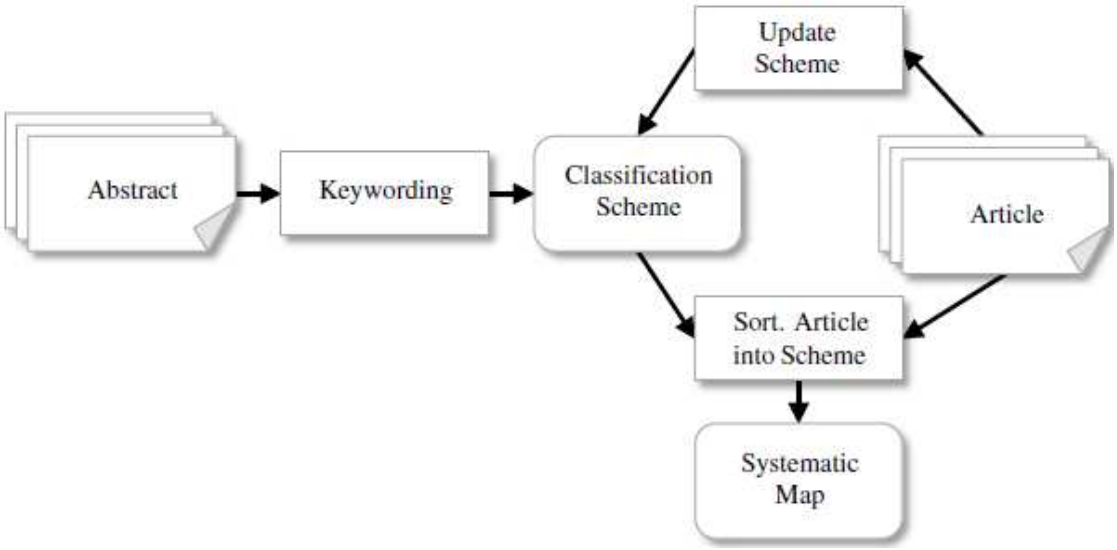

Fig. 4. Classification scheme (adapted from Petersen *et al*. (2008a))

Systematic Mapping Discussion of Results

This section provides a discussion with regard to the obtained results of this SM. Thus, Section 4.1 presents an overview of the retrieved studies based on the results from the application of Filter #1 (Fig. 2) and Sections 4.2 and 4.3 provide graphical representations and discuss the selected studies from Filter #3.

Systematic Mapping Overview

This SM was carried out until May/2017. A total of 2096 primary studies was obtained (no duplicated studies) by applying the proposed search strings to the defined data sources.

Scopus search mechanism retrieved the major studies represented by 827 (33%). IEEE Xplore retrieved 525 studies (21%), whereas ACM Digital Library retrieved 408 studies (16%). In addition, ELSEVIER ScienceDirect retrieved 467 studies (18%) and Compendex 197 studies (8%) (Table 1, column Filter #1). Most of the 433 duplicated studies were retrieved from search engines Compendex, Google Scholar and Scopus. After applying the filters, was selected for fully reading (Filter #3) 11 IEEE Xplore studies, 7 ACM Digital Library studies, 4 ELSEVIER ScienceDirect studies, 2 Scopus studies and 5 Compendex studies (including SCITEPRESS and SpringerLink).

A few studies (3) from Google Scholar were selected for fully reading, due to the fact Google Scholar retrieved 105 studies, including 31 duplicated papers also retrieved from IEEE Xplore and ACM Digital Library. This search mechanism represents around 4% of the total primary studies retrieved. For the final set of 32 selected papers (Filter #3), Google Scholar represents 9.4%.

Selected Studies Discussion

This section presented the obtained results based on Filter #2 for the 86 studies in Table 3. According to the analysis of Filter #2, the research questions of this study were answered by means of the 32 primary studies from Filter #3.

Furthermore, next sections analyze the selected studies in terms of data sources, research types, research questions (RQs), defect types and inspection techniques.

Data Sources x RQs x Research Types

Filter #3 (Fig. 3) resulted in 32 studies selected. It allowed us to answer research questions RQ1 and RQ2 based on data sources and research types (Fig. 5).

Studies from ACM Digital Library, Compendex, IEEE Xplore, ELSEVIER ScienceDirect, Google Scholar and Scopus are related to answer each research question (RQ1 and RQ2). Thus, taking Fig. 5 into account, most of the selected studies are from IEEE Xplore and ACM Digital Library, with 11 and 7 studies, respectively. Defect Types and Software Inspection Techniques: A Systematic Mapping Study 13 IEEE Xplore has six studies that answer RQ1, from which most are classified as Solution Proposal. In addition, RQ2 was answered by means of five studies from ACM Digital Library, classified as Validation Research.

Five studies from Compendex answered RQs, four for RQ1 and one for RQ2. In addition, four studies from ELSEVIER ScienceDirect answered RQs, two for RQ1 and two for RQ2. Furthermore, three studies from Google Scholar answered RQs, two for RQ2 and one for RQ1. Scopus answered once two studies for RQ1.

Table 3. Retrieved studies based on the inclusion and exclusion criteria

Author(s)	Title	Year	Data source
Cox *et al*. (2004b)	A use case description inspection experiment	2004	Google Scholar
Travassos *et al*. (2001)	working with UML: A software design process Based on inspections for the unified modeling language	2001	ELSEVIER ScienceDirect
Amoui *et al*. (2013)	Search-based duplicate defect detection: An industrial experience	2013	ACM Digital Library
Mello *et al*. (2012)	Checklist-based inspection technique for feature models review	2012	IEEE Xplore
Cunha *et al*. (2012)	A set of inspection technique on software product line models	2012	Google Scholar
Mello *et al*. (2010)	Activity diagram inspection on requirements specification	2010	IEEE Xplore
Chen *et al*. (2009)	Variability management in software product lines: A systematic review	2009	ACM Digital Library
Petersen *et al*. (2008b)	The impact of time controlled reading on software inspection effectiveness and efficiency: A controlled experiment	2008	ACM Digital Library
Simidchieva *et al*. (2007)	Representing process variation with a process family	2007	Google Scholar
Winkler *et al*. (2007)	Early software product improvement with sequential inspection sessions: An empirical investigation of inspector capability and learning effects	2007	IEEE Xplore
Tørner *et al*. (2006)	Defects in automotive use cases	2006	ACM Digital Library
He and Carver (2006)	PBR vs. checklist: A replication in the N-fold inspection context	2006	ACM Digital Library
Lange and Chaudron (2006)	Effects of defects in UML models – an experimental investigation	2006	ACM Digital Library
Wagner (2006)	A model and sensitivity analysis of the quality economics of defect-detection techniques	2006	ACM Digital Library
Cooper *et al*. (2005)	Experiences using defect checklists in software engineering education	2005	Google Scholar
Belgamo *et al*. (2005)	TUCCA improving the effectiveness of use case construction and requirement analysis	2005	IEEE Xplore
Staron *et al*. (2005)	An empirical assessment of using stereotypes to improve reading techniques in software inspections	2005	ACM Digital Library
Denger *et al*. (2004)	Investigating the active guidance factor in reading techniques for defect detection	2004	IEEE Xplore
Lanubile *et al*. (2004)	Assessing the impact of active guidance for defect detection: A replicated experiment	2004	IEEE Xplore
Denger and Paech (2004)	an integrated quality assurance approach for use case based requirements	2004	Google Scholar
Grunbacher *et al*. (2003)	An empirical study on groupware support for software inspection meetings	2003	IEEE Xplore
Kelly and Shepard (2003)	An experiment to investigate interacting versus nominal groups in software inspection	2003	ACM Digital Library
Miller and Yin (2003)	Adding diversity to software inspections	2003	IEEE Xplore
Sabaliauskaite *et al*. (2002a)	An experimental comparison of checklist-based reading and perspective-based reading for UML design document inspection	2002	IEEE Xplore
Sabaliauskaite *et al*. (2002b)	An experimental comparison of checklist-based reading and perspective-based reading for UML design document inspection	2002	ACM Digital Library
Anda and Sjøberg (2002)	Towards an inspection technique for use case models	2002	ACM Digital Library
Kelly and Shepard (2001)	A case study in the use of defect classification in inspections	2001	ACM Digital Library
Freimut *et al*. (2001)	Investigating the impact of reading techniques on the accuracy of different defect content estimation techniques	2001	IEEE Xplore
Biffl *et al*. (2001)	Investigating the cost-effectiveness of reinspections in software development	2001	ACM Digital Library
Biffl and Halling (2000)	Software product improvement with inspection – a large-scale experiment on the influence of inspection processes on defect detection in software requirements documents	2000	IEEE Xplore
Travassos *et al*. (1999)	Detecting defects in object-oriented designs: Using reading techniques to increase software quality	1999	ACM Digital Library
Cheng and Jeffery (1996)	Comparing inspection strategies for software requirement specifications	1996	IEEE Xplore
Mishra and Mishra (2009)	Simplified software inspection process in compliance with international standards	2009	ELSEVIER ScienceDirect
Walia and Carver (2009)	A systematic literature review to identify and classify software requirement errors	2009	ELSEVIER ScienceDirect
Munson *et al*. (2006)	Software faults: A quantifiable definition	2006	ELSEVIER ScienceDirect

Table 3. Continue

Sabaliauskaite *et al*. (2004)	Assessing defect detection performance of interacting teams in objectoriented design inspection	2004	ELSEVIER ScienceDirect
Cox *et al*. (2004a)	An experiment in inspecting the quality of use case descriptions	2004	Google Scholar
Hungerford *et al*. (2004)	Reviewing software diagrams: A cognitive study	2004	IEEE Xplore
Thelin *et al*. (2003)	Prioritized use cases as a vehicle for software inspections	2003	IEEE Xplore
Ciolkowski *et al*. (2003)	Software reviews: The state of the practice	2003	IEEE Xplore
Sabaliauskaite *et al*. (2003)	Further investigations of reading techniques for object-oriented design inspection	2003	ELSEVIER ScienceDirect
Biffl (2003)	Evaluating defect estimation models with major defects	2003	ELSEVIER ScienceDirect
Biffl and Halling (2003)	Investigating the defect detection effectiveness and cost benefit of nominal inspection teams	2003	IEEE Xplore
Laitenberger *et al*. (2001)	An internally replicated quasi-experimental comparison of checklist and perspective-based reading of code documents	2001	IEEE Xplore
Laitenberger and DeBaud (2000)	An encompassing life cycle centric survey of software inspection	2000	ELSEVIER ScienceDirect
Laitenberger *et al*. (2000)	An experimental comparison of reading techniques for defect detection in UML design documents	2000	ELSEVIER ScienceDirect
Dunsmore *et al*. (2000)	The role of comprehension in software inspection	2000	ELSEVIER ScienceDirect
Brykczynski (1999)	A survey of software inspection checklists	1999	ACM Digital Library
Porter *et al*. (1998)	Understanding the sources of variation in software inspections	1998	ACM Digital Library
Miller *et al*. (1998)	Further Experiences with Scenarios and Checklists	1998	Google Scholar
Roper *et al*. (1997)	An empirical evaluation of defect detection techniques	1997	ELSEVIER ScienceDirect
Wu *et al*. (2015)	A case study in specification defects detection using statecharts	2015	IEEE Xplore
Valentim *et al*. (2015)	A controlled experiment with usability inspection techniques applied to use case specifications: Comparing the MIT 1 and the UCE techniques	2015	IEEE Xplore
Ma *et al*. (2014)	A defects classification method for aerospace measurement and control software	2014	Compendex
Silva *et al*. (2016)	A field study on root cause analysis of defects in space software	2016	ELSEVIER ScienceDirect
Rawal and Tsetse (2016)	Analysis of bugs in Google security research project database	2016	IEEE Xplore
Kasubuchi *et al*. (2015)	An empirical evaluation of the effectiveness of inspection scenarios developed from a defect repository	2015	IEEE Xplore
Mohammed *et al*. (2015)	An experimental study on detecting semantic defects in object-oriented programs using software reading techniques	2015	ACM Digital Library
Naveed and Ikram (2015)	A Novel checklist: Comparison of CBR and PBR to inspect use case specification	2015	Compendex SpringerLink
Liu *et al*. (2014)	Automatic early defects detection in use case documents	2014	ACM Digital Library
Rocha *et al*. (2015)	Automating test-based inspection of design models	2015	Compendex SpringerLink
Hentschel *et al*. (2016)	Can formal methods improve the efficiency of code reviews?	2016	Compendex SpringerLink
Bjarnason *et al*. (2014)	Challenges and practices in aligning requirements with verification and validation: A case study of six companies	2014	Compendex SpringerLink
Geraldi *et al*. (2015)	Checklist-based inspection of smarty variability models - proposal and empirical feasibility study	2015	Compendex SCITEPRESS
Alshazly *et al*. (2014)	Detecting defects in software requirements specification	2014	ELSEVIER ScienceDirect
Czibula *et al*. (2015)	Detecting software design defects using relational association rule mining	2015	Compendex SpringerLink
Kovalenko *et al*. (2014)	Engineering process improvement in heterogeneous multi-disciplinary environments with defect causal analysis	2014	Compendex SpringerLink
Tang *et al*. (2015)	Enhancing defect prediction with static defect analysis	2015	ACM Digital Library
Singh *et al*. (2016)	Experimental study on feature selection methods for software fault detection	2016	IEEE Xplore
Hamill and Goseva-Popstojanova (2015)	Exploring fault types, detection activities, and failure severity in an evolving safety-critical software system	2015	Compendex SpringerLink
Yousef (2014)	Extracting software static defect models using data mining	2014	ELSEVIER ScienceDirect
Winkler and Biffl (2015)	Focused inspections to support defect detection in automation systems engineering environments	2015	Compendex SpringerLink
Mäntylä and Itkonen (2014)	How are software defects found? The role of implicit defect detection, individual responsibility, documents and knowledge	2014	ELSEVIER ScienceDirect
Albayrak and Carver (2014)	Investigation of individual factors impacting the effectiveness of requirements inspections: A replicated experiment	2014	Compendex SpringerLink
Lopes *et al*. (2015)	MoLVERIC: An inspection technique for MoLIC diagrams	2015	Scopus
Gopinath *et al*. (2014)	Mutations: How close are they to real faults?	2014	IEEE Xplore

Table 3. Continue

Felderer *et al*. (2014)	On the role of defect taxonomy types for testin requirements: Results of a controlled experiment	2014	IEEE Xplore
Rodriguez *et al*. (2014)	Preliminary comparison of techniques for dealing with imbalance in software defect prediction	2014	ACM Digital Library
Femmer *et al*. (2014)	Rapid requirements checks with requirements smells: Two case studies	2014	ACM Digital Library
Yusop *et al*. (2016)	Reporting usability defects: Do reporters report what software developers need?	2016	ACM Digital Library
Cavezza *et al*. (2014)	Reproducibility of environment-dependent software failures: An experience report	2014	IEEE Xplore
Langenfeld *et al*. (2016)	Requirements defects over a project lifetime: An empirical analysis of defect data from a 5-year automotive project at bosch	2016	Compendex SpringerLink
Saito *et al*. (2014)	RISDM: A requirements inspection systems design methodology -perspective-based design of the pragmatic quality model and question set to SRS	2014	IEEE Xplore
Travassos (2014)	Software defects: Stay away from them. Do inspections!	2014	IEEE Xplore
Silva and Vieira (2016)	Software for embedded systems: A quality assessment based on improved ODC taxonomy	2016	ACM Digital Library
Teixeira *et al*. (2015)	Verification of software process line models: A checklist-based inspection approach	2015	Scopus

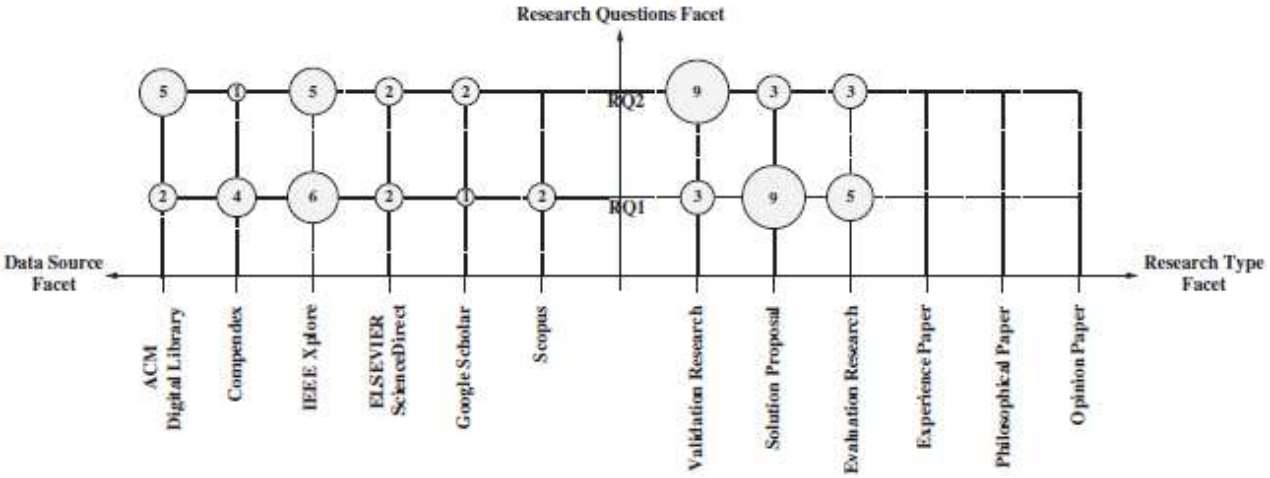

Fig. 5. Answering research questions based on selected studies per data sources and research types (Filter #3)

In an overall analysis, IEEE Xplore and ACM Digital Library provided the most important defect types studies, whereas IEEE Xplore, ACM Digital Library, Compendex, ELSEVIER Science Direct and Google Scholar identified important inspection techniques.

Defect Types x RQs x Research Types

Figure 6 presents the identified defect types, classified according to their research type. We can observe that RQ1 was answered based on the identification of several different and unique defect types. These, were identified mostly as 14 primary studies related with Ambiguities defect type, Inconsistencies (12), Incorrect Facts (10), Omissions (9) and 32 Others defect types amongst 12 of 17 selected studies for RQ1. On the other hand, RQ1 and RQ2, respectively, was answered taking 9 occurrences of research type into account, in which most of the studies are Validation Research and Solution Proposal with several experiments support inspection techniques/approaches.

In addition, based on Fig. 6, requirement engineering provided a means to identify and adapt defect types for several different techniques/approaches. Most of the studies takes IEEE standards as a basis, such as IEEE (1998a). These standards contain recommended practices characterized by establishing a well-defined requirement document, thus providing a means for proposing defect types taxonomies (see Section 4.3).

Taking the word cloud of Fig. 7 into consideration, it highlights the most frequent defect types identified in this study, such as Ambiguities, Inconsistencies and Incorrect Facts. Omissions and Extraneous Information are practically in the same number of occurrence. In overall, the Others unique defect types almost does appears in the same times compared to the Extraneous Information defect type, according to RQ2 (Fig. 6). Remaining defect types barely appear in proposed taxonomies from the selected studies retrieved in this SM.

In the next sections, Fig. 7 and 9 were based on primary studies of Filter #2 and Filter #3.

Defect Types x Software Artifacts

Figure 8 presents the identified defect types based on selected primary studies, classified according to their software artifacts. We can observe that RQ1 was answered based on the identification of number for each defect type and software artifacts related to primary studies. RQ2 is not answered in Fig. 8 due to the fact of being related to studies and several experiments supporting inspection techniques/approaches.

According to Fig. 8, the defect type Ambiguities occurs 14 times compared to the defect types such as Inconsistencies, Incorrect Facts, Omissions and Extraneous Information, which appear approximate. Others different defect types occur 32 times distributed in the primary studies.

As mentioned in this study, the defect types were adapted from requirements engineering. Thus, Fig. 8 presents the defect types with regard to Software Requirements Document (5 times), as well as complemented with Use Cases Diagrams (5 times). In addition, each Class and State UML Diagrams have a frequency below average (2 times) compared to Other Diagrams (6 times).

Related artifacts with Feature Models appear 2 times.

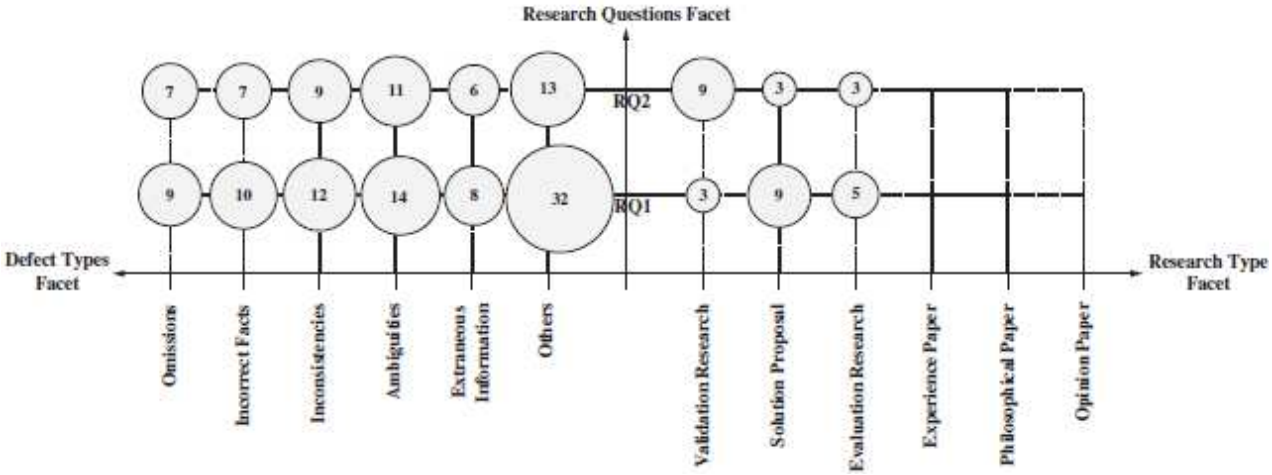

Fig. 6. Answering research questions based on selected studies per defect types and research types (Filter #3)

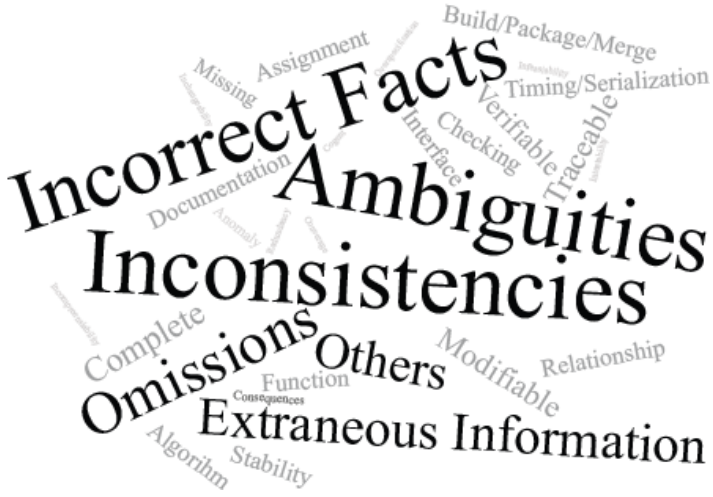

Fig. 7. Word cloud for most frequent defect types

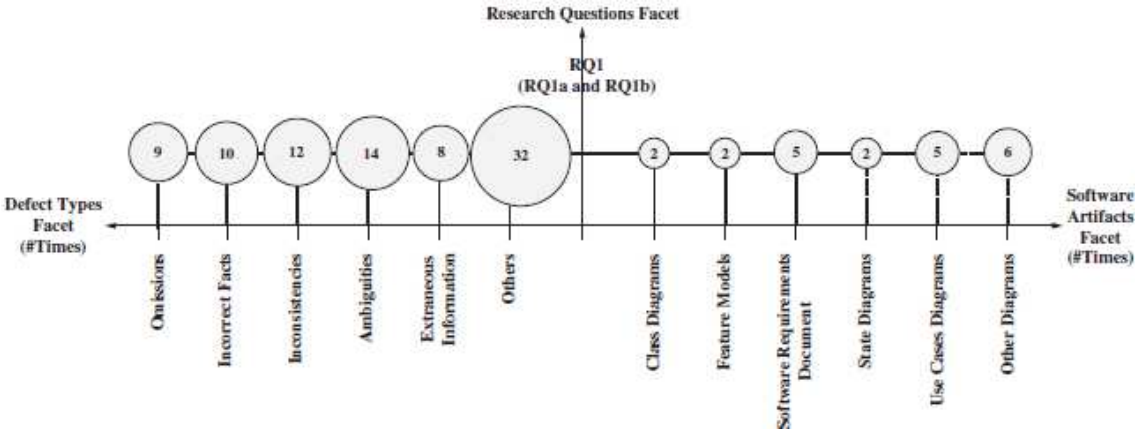

Fig. 8. Answering research question RQ1 based on selected studies per defect types and software artifacts (Filter #3)

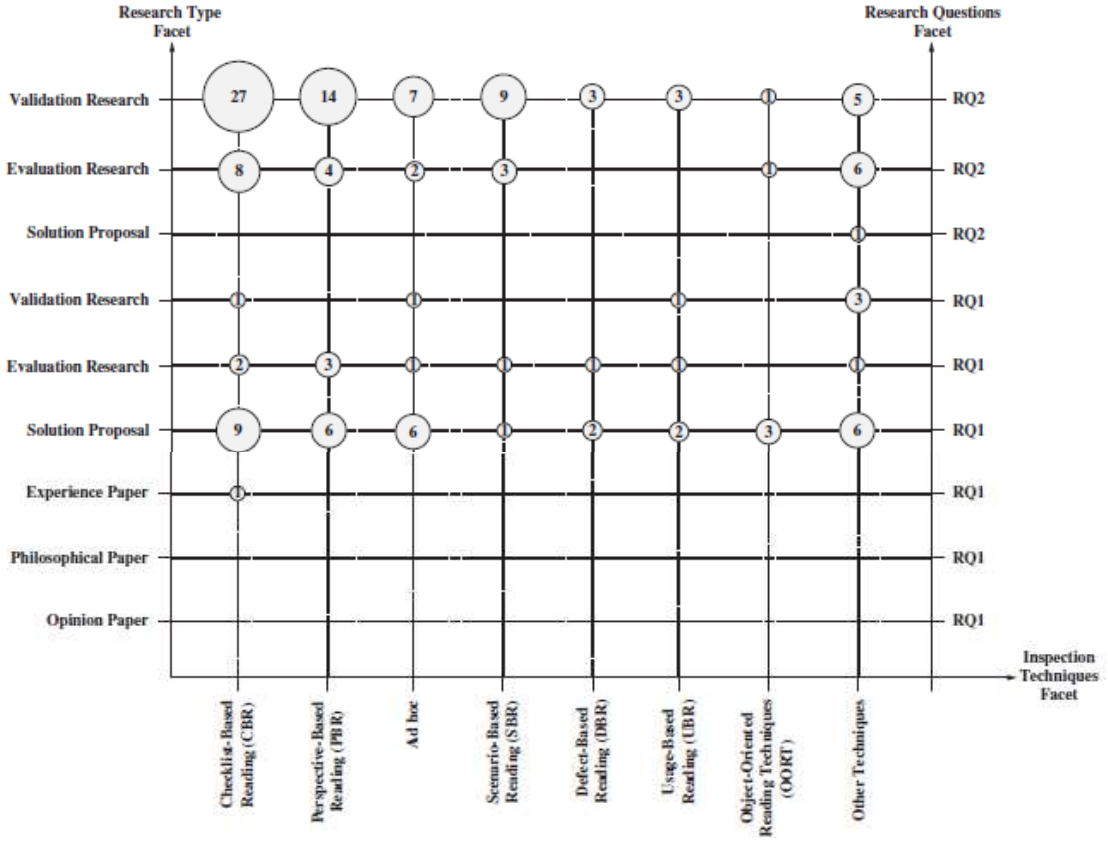

Fig. 9. Software inspection technique types based on the selected studies per RQ and research type

Inspection Technique Type x RQs x Research Types

Figure 9 illustrates the distribution of inspection techniques with relation to research type and RQ1 and RQ2.

By analyzing the studies, we can observe that the majority of software inspection techniques is directly related to Validation Research and Evaluation Research. Therefore, RQ2 was answered by studies related to controlled experiments, empirical studies and case studies. In this sense, RQ1 has few studies with regard to inspection techniques, as its objective is to identify defect types rather than inspection techniques.

According to Fig. 9, the most frequent inspection technique types are: Checklist-Based Reading (CBR) (48 occurrences), Perspective-Based Reading (PBR) (27 occurrences), Ad hoc (17 occurrences) and Scenario-Based Reading (SBR) (14 occurrences).

Different software inspection techniques, other than the ones mentioned, occur with less frequency, such as: Defect-Based Reading (DBR), Usage-Based Reading (UBR) and Object-Oriented Reading Technique (OORT). However, all of them are essential for the research field, as we can see in the discussion presented in next section.

Discussion of the Selected Studies

This section presents a discussion on the final selected papers (32) of this study. Table 4 lists such studies, as well as respective RQs, Authors, Title, Year, Research Type, Data Source, Publication Type and Venue. A discussion is presented based on RQ1 and RQ2, as follows.

Research Question 1 (RQ1a and RQ1b)

Figure 10 presents the results obtained that answered RQ1 with regard to which defect types empirically or non-empirically evidenced. Furthermore, a brief discussion of each study is presented after analyzing the number of times of environments or domains applied such defect types.

Observing the final set of selected studies in the Fig. 10, it is possible highlight that major number of studies (14 of 17) was empirically evidenced by means of experiments, case studies or empirical studies. The three other studies (non-evidenced) not conduct any empirical evaluation.

The main defect types empirically evidenced that appear most times are: Ambiguities (12), Inconsistencies (10), Incorrect Facts (8), Omissions (7), Extraneous Information (6) and Others defect types (10) which are contained in several studies. The non-evidenced defect types appear in equal number of times (2) for three other studies mentioned.

Analyzing the environments or domains applied such defect types empirically evidenced, is possible emphasize that Requirements Engineering domain contains most number of the studies (7). Such common defect types existing in the literature occurs in the studies of Anda and Sjøberg (2002; Belgamo *et al*., 2005; Mello *et al*., 2010; Alshazly *et al*., 2014; Saito *et al*., 2014; Naveed and Ikram, 2015; Langenfeld *et al*., 2016). The Lopes *et al*. (2015) is another study (non-evidenced) that also addresses the Engineering Requirements domain and such defect types. Respectively in this order, a brief discussion of each study of such domain is presented below:

Anda and Sjøberg (2002) proposed a taxonomy for defect types that can be used in checklist-based software inspections, especially for functional requirements and use cases diagrams. Such taxonomy comprises the following defect types (described in Section 2): *Omissions*, *Incorrect Facts*, *Inconsistencies Ambiguities*, *Extraneous Information and Consequences*. In addition anda and Sjøberg (2002) performed two experiments to validate the proposal, providing results to improve such a proposal, as well as evidence of the real usefulness of checklist-based inspections.

The technique proposed by Belgamo *et al*. (2005), named Technique for Use Case Model-based Construction and Construction Requirements Document Analysis (TUCCA), encompasses two different reading techniques, which are different from checklist-based techniques. In addition, the feasibility study conducted by Belgamo *et al*. (2005) provided important results when the TUCCA technique is compared to checklist-based inspection techniques. Thus, defect types that might be applied to inspection techniques based on checklists were identified. Such study mentions the following defect types with relation to the technique used throughout TUCCA requirements to detect defects: *Omission*, *Incorrect Fact*, *Inconsistency*, *Ambiguity and Extraneous Information*.

Mello *et al*. (2010) support the identification of defects in activity diagrams, in which software inspection techniques might be applied. Thus, the research presents a checklist-based technique, as well as its specification of defects. It is concerned on supporting the review of activity diagrams for requirements specification activities. According to the following defect types were adapted from the literature: *Omission*, *Ambiguity*, *Inconsistency*, *Incorrect Fact and Extraneous Information.*

Alshazly *et al*. (2014) investigated problems in the defects detection and common existing inspection techniques in the literature. This work also presents a combined reading technique based on taxonomy to detect different defect types in requirements. The main defect types identified were: Omission, Superfluous, Ambiguous, Inconsistent, Not-conforming to standards, Incorrect, Not-implementable and Risk-prone; and the following techniques: Ad hoc; Requirements Validation Techniques (RVTs); CBR; PBR; SBR; UBR and DBR.

Saito *et al* (2014) proposed RISDM (Requirements Inspection System Design Methodology) a technique set based on the PQM (Pragmatic Quality Model) and PBR. In such paper, it was analyzed more than 140 projects of NTT DATA for five years. Taking into account the analyzed characteristics such study reveals a relationship between cost and level of quality to predicting risks in the maturity of SRS (Software Requirements Specification). The following defect types were identified in such study: *Modifiable*, *Traceable*, *Verifiable*, *Unambiguous*, *Ranked for importance and/or stability*, *Complete and Correct*.

Naveed and Ikram (2015) present an experimental study conducted to identify defects from the UCS (Use Case Specification) and compare CBR and PBR techniques in industry to propose a novel checklist. According to the obtained results, PBR detected more defects, but the effort (person hours) is major compared to CBR. CBR has more efficiency and reports less false positive defects in comparison to PBR. CBR is recommended for medium to small companies. The following defect types were identified in such study: *Ambiguity*, *Incorrectness*, *Inconsistency and Incompleteness*.

Table 4. Final set of primary studies (ordered by RQ)

RQ	Author(s)	Title	Year	Research type	Data source	Publication type	Publication venue
RQ1	Anda and Sjøberg (2002)	Towards an inspection technique for use case models	2002	Solution Proposal	ACM Digital Library	Conference	SEKE
RQ1	Travassos (2014)	Software defects: Stay away from them. Do inspections!	2014	Solution Proposal	IEEE Xplore	Conference	QUATIC
RQ1	Travassos *et al*. (1999)	Detecting defects in object-oriented Designs: Using reading techniques to increase Software quality	1999	Solution Proposal	ACM Digital Library	Conference	OOPSLA
RQ1	Belgamo *et al*. (2005)	TUCCA Improving the effectiveness of use case construction and requirement analysis	2005	Solution Proposal	IEEE Xplore	Conference	ESEM (ISESE)
RQ1	Cunha *et al*. (2012)	A set of inspection technique on software product line models	2012	Solution Proposal	Google Scholar	Conference	SEKE
RQ1	Mello *et al*. (2012)	Checklist-based inspection technique for feature models review	2012	Solution Proposal	IEEE Xplore	Conference	SBCARS
RQ1	Mello *et al* (2010)	Activity diagram inspection on requirements specification	2010	Solution Proposal	IEEE Xplore	Conference	SBES
RQ1	Munson *et al* (2006)	Software faults: A quantifiable definition	2006	Evaluation Research	ELSEVIER ScienceDirect	Journal	ADVENGSOFT
RQ1	Teixeira *et al*. (2015)	Verification of software process line models: A checklist-based inspection approach	2015	Validation Research	Scopus	Conference	CIBSE
RQ1	Geraldi *et al*. (2015)	Checklist-based inspection of Smarty variability models – proposal and empirical feasibility study	2015	Validation Research	Compendex SCITEPRESS	Conference	ICEIS
RQ1	Naveed and Ikram (2015)	A novel checklist: comparison of CBR and PBR to inspect use case specification	2015	Evaluation Research	Compendex SpringerLink	Journal	CCIS
RQ1	Rocha *et al*. (2015)	Automating test-based inspection of design models	2015	Solution Proposal	Compendex SpringerLink	Journal	Softw. Qual. J.
RQ1	Lopes *et al*. (2015)	MoLVERIC: An inspection technique for MoLIC diagrams	2015	Solution Proposal	Scopus	Conference	SEKE
RQ1	Kasubuchi *et al*. (2015)	An empirical evaluation of the effectiveness of inspection scenarios developed from a defect repository	2015	Validation Research	IEEE Xplore	Conference	ICSME
RQ1	Langenfeld *et al*. (2016)	Requirements defects over a project lifetime: An empirical analysis of defect data from a 5-year automotive project at Bosch	2016	Evaluation Research	Compendex SpringerLink	Conference	FSQ
RQ1	Alshazly *et al*. (2014)	Detecting defects in software requirements specification	2014	Evaluation Research	ELSEVIER ScienceDirect	Journal	AEJ
RQ1	Saito *et al*. (2014)	RISDM: A requirements inspection systems design methodology - perspective-based design of the pragmatic quality model and question set to SRS	2014	Evaluation Research	IEEE Xplore	Conference	RE
RQ2	Brykczynski (1999)	A survey of software inspection checklists	1999	Evaluation Research	ACM Digital Library	Journal	ACM SIGSOFT
RQ2	Staron *et al* (2005)	An empirical assessment of using stereotypes to improve reading techniques in software inspections	2005	Validation Research	ACM Digital Library	Conference	WoSQ
RQ2	Denger and Paech (2004)	An integrated quality assurance approach for use case based requirements	2004	Solution Proposal	Google Scholar	Conference	LNI, GI
RQ2	Biffl and Halling (2000)	Software product improvement with inspection	2000	Validation Research	IEEE Xplore	Conference	EUROMICRO
RQ2	Laitenberger *et al*. (2001)	An internally replicated quasi-experimental comparison of checklist and perspective-based reading of code documents	2001	Validation Research	IEEE Xplore	Journal	TSE
RQ2	Sabaliauskaite *et al*. (2004)	Assessing defect detection performance of interacting teams in object-oriented design inspection	2004	Validation Research	ELSEVIER ScienceDirect	Journal	INFSOF (IST)
RQ2	Sabaliauskaite *et al*. (2002a)	An experimental comparison of checklist-based reading and perspective-based reading for UML design document inspection	2002	Validation Research	ACM Digital Library	Conference	ESEM (ISESE)

Table 4. Continue

RQ2	Thelin *et al*. (2003)	Prioritized use cases as a vehicle for software inspections	2003	Validation Research	IEEE Xplore	Journal	IEEE Software
RQ2	Cox *et al*. (2004a)	An experiment in inspecting the quality of use case descriptions	2004	Validation Research	Google Scholar	Journal	JRPIT
RQ2	Mohammed *et al*. (2015)	An experimental study on detecting semantic defects in object-oriented programs using software reading techniques	2015	Validation Research	ACM Digital Library	Conference	ICEMIS
RQ2	Ma *et al*. (2014)	A defects classification method for aerospace measurement and control software	2014	Solution Proposal	Compendex	Conference	ICCEIS
RQ2	Gopinath *et al*. (2014)	Mutations: how close are they to real faults?	2014	Validation Research	IEEE Xplore	Conference	ISSRE
RQ2	Silva and Vieira (2016)	Software for embedded systems: A quality assessment based on improved ODC taxonomy	2016	Solution Proposal	ACM Digital Library	Conference	SAC
RQ2	Silva *et al* (2016)	A field study on root cause analysis of defects in space software	2016	Evaluation Research	ELSEVIER ScienceDirect	Journal	RESS
RQ2	Felderer *et al*. (2014)	On the role of defect taxonomy types for testing requirements: Results of a controlled experiment	2014	Evaluation Research	IEEE Xplore	Conference	SEAA

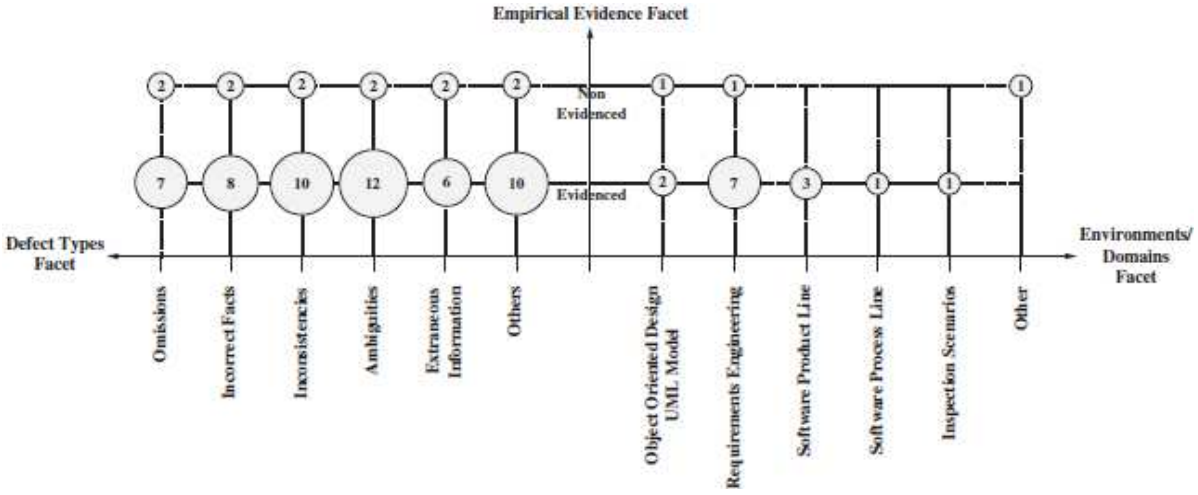

Fig. 10. Answering RQ1 based on empirical evidence of final set of selected studies per defect types and environments/domains (Filter #3)

Langenfeld *et al*. (2016) identified nine defect sources taking in to account requirements defects analyzed 588 requirements reported on an automotive project at Bosch. The most costly and common defect types are incomplete and inconsistency according to refined IEEE 830 standard (IEEE, 1998a). The results obtained to authors allowing decisions improved for the requirements engineering process enabling adopt a new classification of the requirements defects. The following defect types were identified in such study: *Incorrect*, *Ambiguous*, *Incomplete*, *Inconsistent*, *Not ranked*, *Not verifiable*, *Not modifiable and Not traceable*.

Lopes *et al*. (2015) proposed MoLVERIC, a inspection technique for Modeling Language for Interaction Conversation (MoLIC) diagrams applying cards with verification items in the inspections. A pilot study was conducted to verify feasibility and improve such technique. The Travassos *et al*. (2001) taxonomy of defect types was adopted: *Omission*, *Ambiguity*, *Incorrect Fact*, *Inconsistency and Extraneous Information*.

The Object Oriented Design UML Model domain contains 3 studies. Two studies are empirically evidenced and one study is non-evidenced. Thus, such common defect types existing in the literature occurs in the three studies of Travassos *et al*. (1999; Rocha *et al*., 2015; Travassos, 2014) non-evidenced. The next discussions of each study of such domain are presented as follows:

The study performed by Travassos *et al*. (1999) deserves attention for highlighting software defect types related to specific object-oriented modeling. Thus, Travassos *et al*. (1999) present a taxonomy of defect types roughly equivalent to the taxonomy proposed by Anda and Sjøberg (2002). However, Travassos *et al*. (1999) taxonomy is applied to a set of reading techniques, named Traceability-Based Reading (TBR), which was experimentally evaluated in the study. The

defect types applied to the TBR technique are (discussed in Section 2): *Omission*, *Incorrect Fact*, *Inconsistency*, *Ambiguity and Extraneous Information*.

Rocha *et al*. (2015) proposes an automating test-based inspection, named Automated Guided Inspection Technique (AGI), adopting Model-Driven Architecture (MDA) for UML models. The authors of such study conducted three case studies to observe the effectiveness of AGI. The results obtained reveals good coverage of different defect types and defect detection rate per time is similar to the other techniques: PBR; Guided Inspection (MGI) and OORT. These facts are due the AGI not depend on human intervention. The following defect types were identified in this study: *Omission*, *Incorrectness*, *Ambiguity*, *Inconsistency and Extraneous Information*.

Travassos (2014) discusses the benefits of software inspections to evaluate feasibility and effectiveness to support identification of defects existing in most software projects. He presents the main defect types and inspection techniques published in the literature. The defect types are: Omission, Incorrect Fact, Inconsistency, Ambiguity, Extraneous Information and Others; and the techniques: Ad hoc; CBR; SBR; PBR; UBR; OORT and Web Design Perspectives-based Usability Evaluation.

The Software Product Line domain contains 3 studies empirically evidenced. Thus, such common defect types existing in the literature occurs in the studies of Cunha *et al*. (2012; Mello *et al*., 2012; Geraldi *et al*., 2015). A brief discussion of such studies and respective domain is presented as follows:

Cunha *et al*. (2012) proposes a set of inspection techniques, named Software Product Lines Inspection Techniques (SPLIT), aiming at evaluating Software Product Line models. Furthermore, the SPLIT technique takes into consideration Software Requirements Document, Product Map and Feature models for defect detection. In this context, to assess this set of inspection techniques, an experiment was carried out to compare an inspection approach based on defect types to SPLIT. The experimental results favored SPLIT, as a larger amount of defects was found compared to the inspection approach based on defect types. Therefore, the defect types applied using SPLIT technique are: *Redundancies*, *Anomalies and Inconsistencies*.

Mello *et al*. (2012) proposes a checklist-based inspection technique to support the identification of defects in feature models of Software Product Lines, named FMCheck. The main difference between FMCheck and other techniques of software inspection relies on which artifact the checklist is applied to. In FMCheck, the checklist is applied to the feature model, whereas UML models are inspected in different techniques. The following defect types identified using FMCheck were adapted from Travassos *et al*. (1999): *Omission*, *Incorrect Fact*, *Inconsistency*, *Ambiguity and Extraneous Information*.

Geraldi *et al*. (2015) proposes SMartyCheck, a checklist-based software inspection technique for SPL use case and class variability models according to the Smarty approach. The empirical study conducted in such work provides incipient evidence of the SMartyCheck feasibility and improves such technique by means of feedback obtained from several experts. The following defect types are supported by SMarty-Check: *Business Rule*, *Incomplete*, *Inconsistency*, *Incorrect*, *Incorrect Fact*, *Ambiguous*, *Non-modifiable*, *Anomaly*, *Instable*, *Infeasible*, *Omission*, *Extraneous Information and Intentional Deviation*.

The Software Process Line domain contains only the Teixeira *et al* (2015) study empirically evidenced. Such study is discussed as follows: Teixeira *et al* (2015) proposed a preliminary version of PVMCheck, a checklist-based inspection technique to detect defects in Software Process Line (SPrLs) feature models represented using Odyssey-Process-FEX notation. This study was based on experience to previous work of Mello *et al*. (2012) (FMCheck) and provided better understand to literature with regard to inspections in Software Product Line (SPL). Furthermore, PVMCheck was evaluated by means of a quasi-experiment for observing its feasibility compared to Ad hoc technique inspections. The following taxonomy of defect types was adopted: *Omission*, *Incorrect Fact*, *Inconsistency*, *Ambiguity and Extraneous Information*.

The Inspecion Scenarios and Other environments or domains, respectively contains, only the Kasubuchi *et al*. (2015) study empirically evidenced and the Munson *et al*. (2006) study non-evidenced. Such studies are discussed as follows:

Kasubuchi *et al*. (2015) empirically investigated the effectiveness of inspection scenarios developed from a defect repository. These scenarios were investigated throughout of cluster analysis developed based on effectiveness. Based on obtained results, nine distinct defects were identified corresponding inspection scenarios. Furthermore, inspection scenarios also can be obtained by the checklist proposed as: Value-Based Review (VBR). The following defect types were identified in such study: *Omitted*, *Ambiguous*, *Incorrect*, *Insufficient and Misleading*; and the following techniques: VBR and UBR.

On the other hand, compared to the previously mentioned studies, the research conducted by Munson *et al*. (2006) performs attempts to quantify software failures by taking defect as a special type of failure based on evaluation of failures throughout a specific software grammar.

In summary, the studies presented in this section answered RQ1 and provided defect types taking into account by software inspection techniques, encompassing studies such as: (i) new adapted taxonomies for detect defect types and different artifacts (Anda and Sjøberg, 2002; Travassos *et al*.,

1999); (ii) new inspection techniques evaluated by experiments (e.g., TUCCA (Belgamo *et al*., 2005), SPLIT (Cunha *et al* 2012), FMCheck (Mello *et al*., 2012), PVMCheck (Teixeira *et al*., 2015; SmartyCheck (Geraldi *et al*., 2015), AGI (Rocha *et al*., 2015) and MoLVERIC (Lopes *et al*., 2015); (iii) experiments that presented defect types based on requirements engineering and compared inspection techniques (Alshazly *et al*., 2014; Kasubuchi *et al*., 2015; Langenfeld *et al*., 2016); and (iv) an overview of existing defect types by Travassos (2014) and emphasis on failures by Munson *et al*. (2006).

Research Question 2 (RQ2a and RQ2b)

Figure 11 presents the results obtained that answered RQ2 with regard to which inspection techniques/approaches provide evidence of defect types and has documented evaluation results. A brief discussion of each study is presented after analyzing the number of times of software inspection techniques/approaches applied such defect types.

Observing the final set of selected studies in the Figure 11, it is important highlight that all number of inspection techniques/approaches was documented by means of experiments, case studies or empirical studies. Furthermore, the main defect types evidenced by means of software inspection techniques/approaches that occurs most times in studies are: Ambiguities (11), Inconsistencies (9), Incorrect Facts and Omissions (7), Extraneous Information (6) and Others defect types (13) which are contained in several studies.

Analyzing the software inspection techniques/approaches applied such defect types, is possible emphasize that CBR technique contains most number of the studies (10), PBR (6), SBR (1), Ad hoc (1) and Other Techniques (6) in several studies. Thus, such common defect types existing in the literature occurs in the studies of Brykczynski (1999; Biffl and Halling, 2000; Laitenberger *et al*., 2001; Sabaliauskaite *et al*., 2002a; 2004; Thelin *et al*., 2003; Denger and Paech, 2004; Cox *et al*., 2004a; Staron *et al*., 2005; Mohammed *et al*., 2015). A brief discussion of each study of such techniques/approaches is presented as follows:

Research conducted by Brykczynski (1999) is aimed at identifying which items of checklists developed by Fagan (1976; 1986) between the 70's and 80's, are considered important to software inspection processes. Thus, it was analyzed 117 checklists from 24 different sources in order to validate the research. Therefore, it allowed observing that conducting inspections by means of checklist-based reading technique is effective. In addition, this technique is widely adopted by industry when inspecting artifacts based on checklists in different software contexts.

The study performed by Biffl and Halling (2000) experimentally investigated the effect of CBR, SBR and PBR reading techniques and detection of defects involved in quality inspection process. Therefore, obtained results corroborated the effectiveness of such techniques. The CBR technique obtained better effectiveness results than PBR in most cases.

The quasi-experiments performed by Laitenberger *et al*. (2001) also established comparisons to identify the effectiveness and cost of defects detection in the PBR technique with respect to CBR. In order to evaluate this scenario, two replications were carried out taking practitioners from Bosch Telecom GmbH into consideration. Then, overall obtained results provided evidence that PBR is more effective than CBR. PBR technique reduced the cost per defect found and contributed to the detection of a larger amount of defects during inspections meetings. In addition, it achieved lower costs for identifying defects based on the effort of the subjects.

Sabaliauskaite *et al*. (2002a) evaluated the performance of two inspection groups throughout experiments using two reading techniques: CBR and PBR. In this scenario, obtained results from comparing these techniques reveal no significant differences between them.

Another study conducted by Sabaliauskaite *et al*. (2004) also compared the CBR and PBR techniques, but in a different context from their previous study (Sabaliauskaite *et al*., 2002a). It presents an evaluation of inspected UML documents design. The following results were obtained: Similar effectiveness in detecting defects in both inspection techniques; reviewers who used PBR spent less time inspecting artifacts than CBR reviewers; and cost per defects found using CBR is less than that using PBR.

Another important experimental study conducted by Thelin *et al* (2003) presents the Usage-Based Reading (UBR) technique, which was experimentally evaluated by comparing it with the CBR technique. It was analyzed by means of three tests, which evaluated the efficiency (defects found per hour), efficacy (percentage of defects found) and preparation (inspection time in minutes). Thus, the experiment indicated that the UBR technique is significantly more efficient and effective than the CBR technique.

The study of Denger and Paech (2004) presents an integrated approach to ensure use case quality. Such an approach combines use case guidelines and inspection techniques for use cases. The approach is evaluated by means of simulations. Thus, the proposed combination is performed based on defects classification and classes of defects, which were identified taking quality criteria into account. Denger and Paech (2004) only evaluated CBR and PBR techniques; thus, both were practically equal in detecting defects. Therefore, by means of experimentation, obtained results provided initial evidence on increasing efficiency and effectiveness to ensure quality of use cases.

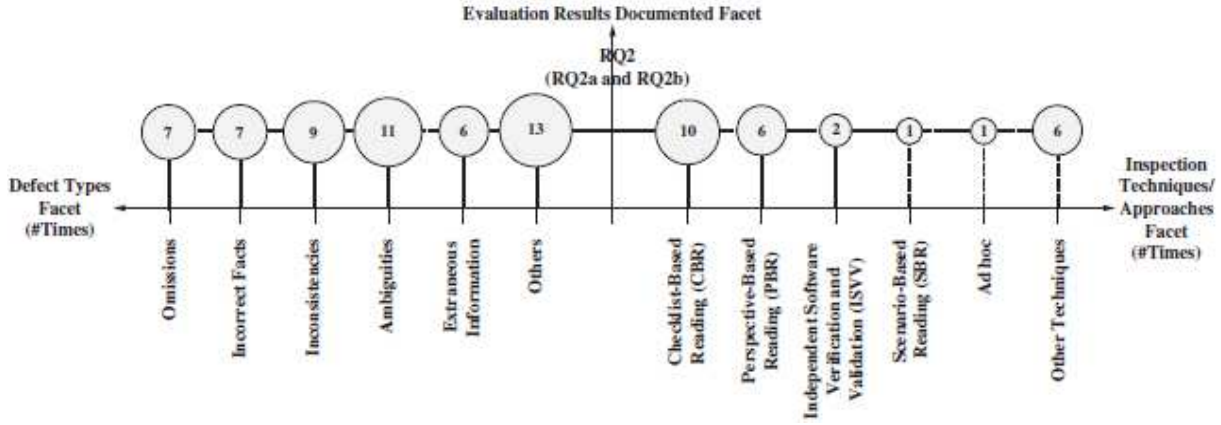

Fig. 11. Answering RQ2 based on evaluation results documented of final set of selected studies per defect types and inspection techniques/approaches (Filter #3)

Cox *et al*. (2004a) presented experimental results at the application of a checklist-based inspection to the technical process of use cases description or specification. Such an experiment compared a checklist-based technique to an Ad hoc technique based on a group of subjects, who overall identified more defects applying the checklist-based technique.

It is important to highlight the study conducted by Staron *et al*. (2005), which provided evidence, based on several experiments, that the stereotypes present in UML models contribute to ensure software inspection process quality, as well as the reduction of defects by applying CBR techniques and PBR. Thus, obtained results evidenced that the CBR technique is more efficient and PBR is more effective. Therefore, stereotypes are essential to ensure quality of the carried out software inspections.

Mohammed *et al*. (2015) present a controlled experimental study to compare the effectiveness and efficiency of three reading techniques: CBR; Functional-Based Reading (FBR) and Systematic-Based Reading based on an approach to discovering semantic defects in object oriented programming. The results of such study showed that FBR technique is more productive and effective than the Systematic-Based Reading and CBR.

The Independent Software Verification and Validation (ISVV) approach occurs 2 times (2 studies) in the studies of Silva and Vieira (2016; Silva *et al*., 2016). Thus, a brief discussion of such studies is presented as follows:

Silva and Vieira (2016) proposed adaptations to the ODC taxonomy applied to a real dataset with 1070 ISVV issues selected and classified from space critical systems. The results obtained from this new classification scheme allowed to analyze the classification gaps, clustering and reclassifying this dataset. Thus, in another study, Silva *et al*. (2016) presented a field study applying an improved ODC and ISVV in four critical space software projects, detecting 1070 defects through root cause analysis. According to the method of their work can also prevent defects and optimize V&V activities throughout a proposal of a generic process that suggest corrections of defects. The following adaptations were applied in ODC: *Traceability/ Compatibility*, *Consistency/Completeness*, *Standards conformance*, *Rare situation*, *White box path coverage*, *HW/SW configuration*, *Build/Package/Environment and Interface*.

The Others Techniques occurs 6 times in 4 studies. Thus, such common defect types existing in the literature occur in the studies of based on techniques: Fault Injection Technique in Gopinath *et al*. (2014), Sequence-oriented Test Design (TS) and Performance-oriented Test Design (TP) in Felderer *et al*. (2014), UBR in Thelin *et al* (2003) and Systematic-Based Reading and FBR in Mohammed *et al*. (2015). Thus, a brief discussion of such studies of such techniques/approaches is presented as follows:

Gopinath *et al*. (2014) attempted identify real faults in mutations analysis applying patches in different programming languages (Java, C, Python, Haskell) for detect distinct bugs. Based on results obtained, mutation operators are not representative of real faults. Furthermore, Competent Programmer Hypothesis is not applicable in this scenario. The authors mentioned that ODC is not inappropriate in the mutation analysis. However, ODC included functional, algorithmic or serialization errors.

Felderer *et al*. (2014) investigated and conducted a controlled experiment with students to compare the influence of two types of top-level defect categories to possibly create new defects taxonomies. The results obtained tested requirements as well as their consequences in an industrial application. Researchers can use this study as a basis for proposing new taxonomies.

No technique was mentioned or discussed in the study of Ma *et al* (2014). The authors present a systematic classification method for aerospace measurement and control software defects. The defect

types of proposed classification method were adapted from classification methods in the literature as: Orthogonal Defect Classification (ODC); Putnam classification method; Thayer classification method; and IEEE Std. 1044-1993. Taking into account the detection of specific defect types, this classification method can corroborate with the improvement of the stability and effectiveness of the tests of aerospace measurement and control software.

In summary, the studies presented in this section answered RQ2 and provide evidence of the existing inspection techniques or approaches associated with defect types, encompassing studies, such as: (i) several academic validation studies that compare and evaluate the effectiveness of the main inspection techniques (e.g., CBR, PBR, SBR, Ad hoc, among others) in distinct contexts (Biffl and Halling, 2000; Cox *et al* 2004a; Denger and Paech, 2004; Laitenberger *et al*., 2001; Mohammed *et al*., 2015; Sabaliauskaite *et al*., 2002a; 2003; Staron *et al*., 2005; Thelin *et al*., 2003); (ii) case studies in industry (Felderer *et al*., 2014; Ma *et al*., 2014, Silva and Vieira 2016, Silva *et al*., 2016); and (iii) application and adaptation of systematic classifications such as ODC (Gopinath *et al*., 2014; Ma *et al* 2014; Silva and Vieira 2016; Silva *et al* 2016).

Results Overview of the Selected Studies

In an overall analysis, the diversity of study types is evident as they encompass: (i) Proposals of taxonomies for defect types, experimentally evaluated for specific UML models; (ii) different proposals and comparisons of techniques or approaches for software inspection and detection of defects; (iii) several experiments taking different software inspection techniques into consideration; and UML models quality evaluation studies by means of stereotypes that may be related to software inspection techniques.

Interestingly, the retrieved and selected studies in this SM are presented in a motivating perspective. Amongst the discussed studies we observed distinct researches and issues in several scenarios, mostly experimental. Therefore, this carried out SM is considered essential to identify studies that might mitigate any kind of support when detecting defects in software inspection techniques.

It is important to emphasize the taxonomies and classification of defect types usually adapted for detecting defects based on requirements engineering. Thus, the defect types addressed in this SM can be used independent both in inspection techniques and different models (e.g., UML diagrams).

Amongst the main gaps identified between the primary studies, the following are highlighted: (i) several inspection techniques and respective defect types do not encompass all existing UML models; (ii) taxonomies do not propose new defect types or adapt all the main defect types in the literature; (iii) several software engineering artifacts are not inspected in software product lines or process (e.g., Component or Activity SPL UML diagrams, BPM (Business Process Management), among others); and (iv) inspection techniques and taxonomies could be adapted or improved to specific contexts, such as, agile methodology or PBR could be adapted to inspect artifacts in security network systems.

As several experiments have been identified from the discussed studies, they contribute to the understanding of the existing defect types and inspection techniques in the literature. They provide essential evidence for evolving inspection techniques by identifying, adapting and applying them to different defect types and contexts.

Threats to Validity

The following major threats to the validity of this study are discussed as follows:

Research Questions

Two research questions and sub-questions were defined for conducting this study. Such research questions were analyzed and refined before being finally drawn. We believe that more generic keywords returned more important primary studies to the scope of this study. However, this study was focused only on defect types and software inspection techniques.

Data Sources

Four data sources were selected, which have been considered essential for the software engineering community based on several performed systematic mappings and literature reviews. However, the more search sources are taken into account, the larger the set of selected studies might potentially be. Prospective secondary studies should consider expanding the data source list.

Publication Bias

We cannot guarantee that all primary studies related to this SM were retrieved. This issue may occur due to the fact that the defined data source engines are not as precise as we desire for processing and executing queries based on the defined keywords.

Unfamiliarity with Other Fields

The improvement of the keywords and search strings defined for this study could aid and optimize searching the most important primary studies in other research areas. For instance, primary studies of the information security area could be investigated in order to identify what defect types occur in computer networks infrastructure, taking defects present in simpler and more complex networks into account. Therefore, possible defect types present in such an area were not considered in this study.

Conclusion

The main motivation for developing this study was mapping existing software inspection defect types in the literature aiming at providing taxonomies and classifications to guide researchers and practitioners conducting studies in this topic. Thus, by means of this systematic mapping, new research can be performed in order to evolve existing defect types and software inspection techniques and/or proposing novel approaches for inspecting software in several different contexts, including industrial sets.

We adopted consolidated systematic guidelines (procedures and criteria) presented by the literature in this work to conduct this systematic mapping. Such guidelines helped to organize and present an overview of quantitative results obtained. In addition, these procedures allowed characterize this SM and limited the scope of the discussion of the primary studies simply.

The criteria used to determine the final set of the most important defect types is related to the frequency (number of times) they occur in primary studies selected through the inclusion and exclusion criteria. These defect types are arising from requirements engineering, which have been adapted for different domains (e.g., Software Requirements Document, UML diagrams and Feature Model).

At the time this study was conducted, the existing literature on this research topic lacked studies on defect detection considering several different contexts, such as model-based inspection of software product lines, reference architectures, model-driven architecture, system-of-systems and human-computer interaction. Thus, this SM provides a positioning and guidance to other researchers and practitioners from different areas that may consult the studies presented at the prospect of adopting them in their research, as well as their defect types. Such scenario allows proposing works that integrate areas such as: embedded systems, distributed systems and computer networks security.

By taking the obtained results of this study into account, we provide evidence that most of the mapped studies are related to different contexts, such as experiments, proposals of new inspection techniques and new taxonomies of defect types. Studies have been conducted by large companies (IBM, NASA, JPL, AT&T, Motorola, Nortel, Allianz, Bosch and others) reporting that well-planned software inspections with respective defect types adapted from the literature can contribute to the detection of defects on the average of 80%, as well as improving software quality.

Furthermore, the results may help practitioners to develop new products or automate support tools throughout the use of the main defect types based on taxonomies in the literature addressed by this systematic mapping. We believe this contribution allows industry to save time and money immediately for possible further research.

Once this SM was carried out, we expect that the obtained results might guide future work with an emphasis on the already identified defect types and software inspection techniques, as well as novel studies. Furthermore, mitigate threats to validity by extending unanswered possible research questions and their derivatives considering an improvement in the inclusion and exclusion criteria. In addition, new SMs might be performed taking different data sources and keywords into consideration.

Acknowledgment

The authors would like to thank CAPES/Brazil for supporting this study.

Author's Contributions

Ricardo Theis Geraldi: Contributed in the protocol design, execution and analysis and interpretation and sharing of this empirical study.

Edson OliveiraJr: Supported Ricardo T. Geraldi at designing and filtering primary studies, as well as analysis and interpretation of such studies. Both authors equally contributed at writing and reviewing this paper.

Ethics

We declare no conflict of interest with relation to the primary studies recovered in this systematic mapping. In addition, as far as we know, no study in this mapping contains ethical insufficiency.

References

Albayrak, Ö and J.C. Carver, 2014. Investigation of individual factors impacting the effectiveness of requirements inspections: A replicated experiment. Empirical Software Eng., 19: 241-266.

Alshazly, A.A., A.M. Elfatatry and M.S. Abougabal, 2014. Detecting defects in software requirements specification. Alexandria Eng. J., 53: 513-527. DOI: 10.1016/j.aej.2014.06.001

Amoui, M., N. Kaushik, A. Al-Dabbagh, L. Tahvildari and S. Li *et al*., 2013. Search-based duplicate defect detection: An industrial experience. Proceedings of the 10th IEEE Working Conference on Mining Software Repositories, May 18-19, IEEE Xplore Press, San Francisco, pp: 173-182. DOI: 10.1109/MSR.2013.6624025

Anda, B. and D.I.K. Sjøberg, 2002. Towards an inspection technique for use case models. Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering, Jul. 15-19, ACM Press, Ischia, Italy, pp: 127-134. DOI: 10.1145/568760.568785

Bailey, J., D. Budgen, M. Turner, B.A. Kitchenham and P. Brereton *et al*., 2007. Evidence relating to object-oriented software design: a survey. Proceedings of the 1st International Symposium on Empirical Software Engineering and Measurement, Sep. 20-21, IEEE Xplore Press, Madrid, Spain, pp: 482-484. DOI: 10.1109/ESEM.2007.58

Barney, S., K. Petersen, M. Svahnberg, A. Aurum and H. Barney, 2012. Software quality trade-offs: A systematic map. Inform. Software Technol., 54: 651- 662. DOI: 10.1016/j.infsof.2012.01.008

Belgamo, A., S. Fabbri and J.C. Maldonado, 2005. TUCCA: Improving the effectiveness of use case construction and requirement analysis. Proceedings of the International Symposium on Empirical Software Engineering, Nov. 17-18, IEEE Xplore Press, Noosa Heads, Qld., Australia, pp: 257-266. DOI: 10.1109/ISESE.2005.1541835

Biffl, S. and M. Halling, 2000. Software product improvement with inspection. A large-scale experiment on the influence of inspection processes on defect detection in software requirements documents. Proceedings of the 26th Euromicro Conference, Sep. 5-7, IEEE Xplore Press, Maastricht, pp: 262-269. DOI: 10.1109/EURMIC.2000.874427

Biffl, S. and M. Halling, 2003. Investigating the defect detection effectiveness and cost benefit of nominal inspection teams. Trans. Software Eng., 29: 385-397. DOI: 10.1109/TSE.2003.1199069

Biffl, S., 2003. Evaluating defect estimation models with major defects. J. Syst. Software, 65: 13-29. DOI: 10.1016/S0164-1212(02)00025-0

Biffl, S., B. Freimut and O. Laitenberger, 2001. Investigating the cost-effectiveness of reinspections in software development. Proceedings of the 23rd International Conference on Software Engineering, May 19-19, IEEE Xplore Press, Toronto, pp: 155-164. DOI: 10.1109/ICSE.2001.919090

Bjarnason, E., P. Runeson, M. Borg, M. Unterkalmsteiner and E. Engstr¨om *et al*., 2014. Challenges and practices in aligning requirements with verification and validation: a case study of six companies. Empirical Software Eng., 19: 1809-1855.

Boehm, B. and V.R. Basili, 2001. Software defect reduction top 10 list. J. Comput., 34: 135-137. DOI: 10.1109/2.962984

Brykczynski, B., 1999. A survey of software inspection checklists. Software Eng. Notes, 24: 82-89. DOI: 10.1145/308769.308798

Cavezza, D.G., R. Pietrantuono, S. Russo, J. Alonso and K.S. Trivedi, 2014. Reproducibility of environment-dependent software failures: An experience report. Proceedings of the 25th International Symposium on Software Reliability Engineering, Nov. 3-6, IEEE Xpolore Press, Naples, Italy, pp: 267-276. DOI: 10.1109/ISSRE.2014.19

Chen, L., M.A. Babar and N. Ali, 2009. Variability management in software product lines: A systematic review. Proceedings of the 13th International Software Product Line Conference, Aug. 24-28, Carnegie Mellon University, San Francisco, pp: 81-90.

Cheng, B. and R. Jeffery, 1996. Comparing inspection strategies for software requirement specifications. Proceedings of the Australian Software Engineering Conference, Jul. 14-18, IEEE Xplore Press, Melbourne, pp: 203-211. DOI: 10.1109/ASWEC.1996.534137

Ciolkowski, M., O. Laitenberger abd S. Biffl, 2003. Software reviews: The state of the practice. IEEE Software, 20: 46-51.

Cooper, K., S. Liddle abd S. Dascalu, 2005. Experiences using defect checklists in software engineering education. Proceedings of the International Conference on Computer Applications in Industry and Engineering, Nov. 9-11, Sheraton Moana Surfrider, Honolulu, Hawaii, USA, pp: 402-409.

Cox, K., A. Aurum and R. Jeffery, 2004a. An experiment in inspecting the quality of use case descriptions. J. Res. Pract. Inform. Technol., 36: 211-229.

Cox, K., R. Jeffery and A. Aurum, 2004b. A use case description inspection experiment. University of New South Wales - School of Computer Science and Engineering, Australia.

Cunha, R., T.U. Conte, E.S. Almeida and J.C. Maldonado, 2012. A set of inspection techniques on software product line models. Proceedings of the 24th International Conference on Software Engineering and Knowledge Engineering, Jul. 1-3, Hotel Sofitel, Redwood City, San Francisco Bay, USA, pp: 657-662.

Czibula, G., Z. Marian and I.G. Czibula, 2015. Detecting software design defects using relational association rule mining. Knowledge Inform. Syst., 42: 545-577.

Denger, C. and B. Paech, 2004. An integrated quality assurance approach for use case based requirements. GI-Edition-Lecture Notes in Informatics (LNI) pp: 59-74.

Denger, C., M. Ciolkowski and F. Lanubile, 2004. Investigating the active guidance factor in reading techniques for defect detection. Proceedings of the International Symposium on Empirical Software Engineering, Aug. 20-20, IEEE Xplore Press, Redondo Beach, pp: 219-228. DOI: 10.1109/ISESE.2004.1334909

Dunsmore, A., M. Roper and M. Wood, 2000. The role of comprehension in software inspection. J. Syst. Software, 52: 121-129. DOI: 10.1016/S0164-1212(99)00138-7

Fagan, M., 2002. A History of Software Inspections. 1st Edn., Springer, Berlin Heidelberg.

Fagan, M.E., 1976. Design and code inspections to reduce errors in program development. IBM Syst. J., 15: 182-211. DOI: 10.1147/sj.153.0182

Fagan, M.E., 1986. Advances in software inspections. IEEE Trans. Software Eng., SE-12:744-751. DOI: 10.1109/TSE.1986.6312976

Felderer, M., A. Beer and B. Peischl, 2014. On the role of defect taxonomy types for testing requirements: Results of a controlled experiment. Proceedings of the 40th EUROMICRO Conference on Software Engineering and Advanced Applications, Aug. 27-29, IEEE Xplore Press, Verona, Italy pp: 377-384. DOI: 10.1109/SEAA.2014.37

Femmer, H., D.M. Fernández, E. Juergens, M. Klose and I. Zimmer *et al*., 2014. Rapid requirements checks with requirements smells: Two case studies. Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering, Jun. 03-03, ACM, Hyderabad, India, pp: 10-19. DOI: 10.1145/2593812.2593817

Freimut, B., O. Laitenberger and S. Biffl, 2001. Investigating the impact of reading techniques on the accuracy of different defect content estimation techniques. Proceedings of the 7th International Software Metrics Symposium, Apr. 4-6, IEEE Xplore Press, London, pp: 51-62. DOI: 10.1109/METRIC.2001.915515

Geraldi, R.T., E. Oliveira Jr, T. Conte and I. Steinmacher, 2015. Checklist-based inspection of smarty variability models - proposal and empirical feasibility study. Proceedings of the 17th International Conference on Enterprise Information Systems, Apr. 27-30, SCITEPRESS, Science and Technology Publications, Barcelona, Spain, pp: 268-276. DOI: 10.5220/0005350102680276

Gilb, T. and D. Graham, 1993. Software Inspection. 5th Edn., Addison-Wesley Longman Publishing Co., Inc. Boston, ISBN-10: 0201631814, pp: 496.

Gopinath, R., C. Jensen and A. Groce, 2014. Mutations: How close are they to real faults? Proceedings of the 25th International Symposium on Software Reliability Engineering, Nov. 3-6, IEEE Xplore Press, Naples, Italy, pp: 189-200. DOI: 10.1109/ISSRE.2014.40

Grunbacher, P., M. Halling and S. Biffl, 2003. An empirical study on groupware support for software inspection meetings. Proceedings of the 18th IEEE International Conference on Automated Software Engineering, Oct. 6-10, IEEE Xplore Press, Montreal, Que., pp: 4-11. DOI: 10.1109/ASE.2003.1240289

Hamill, M. and K. Goseva-Popstojanova, 2015. Exploring fault types, detection activities and failure severity in an evolving safety-critical software system. Software Quality J., 23: 229-265. DOI: 10.1007/s11219-014-9235-5

Hayes, J., I. Raphael, E. Holbrook and D. Pruett, 2006. A case history of international space station requirement faults. Proceedings of the 11th International Conference on Engineering of Complex Computer Systems, Aug. 15-17, IEEE Xplore Press, Stanford, CA, pp: 1-10. DOI: 10.1109/ICECCS. 2006.1690351

He, L. and J. Carver, 2006. PBR Vs. checklist: A replication in the N-fold inspection context. Proceedings of the International Symposium on Empirical Software Engineering, Sep. 21-22, ACM, Rio de Janeiro, Brazil, pp: 95-104. DOI: 10.1145/1159733.1159750

Hentschel, M., R. Hähnle and R. Bubel, 2016. Can formal methods improve the efficiency of code reviews? Proceedings of the International Conference on Integrated Formal Methods, Jun. 1-5, Springer, Reykjavik, Iceland, pp: 3-19. DOI: 10.1007/978-3-319-33693-0_1

Hernandes, E.M., A. Belgamo and S. Fabbri, 2013. Experimental studies in software inspection process - a systematic mapping. Proceedings of the International Conference on Enterprise Information Systems, (EIS'13), SciTePress - Science and and Technology Publications, pp: 66-76. DOI: 10.5220/0004454000660076

Hungerford, B., A. Hevner and R. Collins, 2004. Reviewing software diagrams: A cognitive study. Trans. Software Eng., 30: 82-96. DOI: 10.1109/TSE.2004.1265814

IEEE, 2012. System and software verification and validation. Standard 1012-2012, IEEE.

IEEE, 1998a. Recommended practice for software requirements specifications. Standard 830-1998, IEEE.

IEEE, 1998b. Software reviews. Standard 1028-1997, IEEE.

Kalinowski, M. and G.H. Travassos, 2004. A computational framework for supporting software inspections. Proceedings of the 19th International Conference on Automated Software Engineering, Sep. 24-24, IEEE Xplore Press, Linz, Austria, pp: 46-55, DOI: 10.1109/ASE.2004.1342723

Kasubuchi, K., S. Morisaki, A. Yoshida and C. Ogawa, 2015. An empirical evaluation of the effectiveness of inspection scenarios developed from a defect repository. Proceedings of the International Conference on Software Maintenance and Evolution, Sep. 29-Oct. 1, IEEE Xplore Press, Bremen, Germany, pp: 439-448. DOI: 10.1109/ICSM.2015.7332495

Kelly, D. and T. Shepard, 2003. An experiment to investigate interacting versus nominal groups in software inspection. Proceedings of the 2003 Conference of the Centre for Advanced Studies on Collaborative Research, Oct. 06-09, IBM Press, Ontario, Canada, pp: 122-134.

Kelly, D., T. Shepard, 2001. A case study in the use of defect classification in inspections. Proceedings of the conference of the Centre for Advanced Studies on Collaborative Research, Nov. 5-7, IBM Press, Toronto, Ontario, Canada, pp: 5-7. DOI: 10.1145/782096.782103

Kitchenham, B.A., 2007. Guidelines for performing systematic literature reviews in software engineering. Proceedings of the International Conference on Software Engineering, ACM Press, pp: 1051-1052. DOI 10.1145/1134285.1134500

Kitchenham, B.A., D. Budgen and O.P. Brereton, 2010. Using mapping studies as the basis for further research - a participant-observer case study. Inform. Software Technol., 53: 638-651. DOI: 10.1016/j.infsof.2010.12.011

Kovalenko, O., D. Winkler, M. Kalinowski. E. Serral and S. Biffl, 2014. Engineering process improvement in heterogeneous multi-disciplinary environments with defect causal analysis. Commun. Comput. Inform. Sci., 425: 73-85.

Laitenberger, O. and J.M. DeBaud, 2000. An encompassing life cycle centric survey of software inspection. J. Syst. Software, 50: 5-1. DOI: 10.1016/S0164-1212(99)00073-4

Laitenberger, O., C. Atkinson, M. Schlich and K.E. Emam, 2000. An experimental comparison of reading techniques for defect detection in fUMLg design documents. J. Syst. Software, 53: 183-204. DOI: 10.1016/S0164-1212(00)00052-2

Laitenberger, O., K. El Emam and T.G. Harbich, 2001. An internally replicated quasi- experimental comparison of checklist and perspective based reading of code documents. IEEE Trans. Software Eng., 27: 387-421. DOI: 10.1109/32.922713

Lange, C.F.J. and M.R.V. Chaudron, 2006. Effects of defects in UML models: An experimental investigation. Proceedings of the 28th International Conference on Software Engineering, May 20-28, ACM, Shanghai, China, pp: 401-411. DOI: 10.1145/1134285.1134341

Langenfeld, V., A. Post and A. Podelski, 2016. Requirements defects over a project lifetime: An empirical analysis of defect data from a 5-year automotive project at Bosch. Proceedings of the International Working Conference on Requirements Engineering: Foundation for Defect Types and Software Inspection Techniques: a Systematic Mapping Study 33 Software Quality, Springer, pp: 145-160.

Lanubile, F., T. Mallardo, F. Calefato, C. Denger and M. Ciolkowski, 2004. Assessing the impact of active guidance for defect detection: A replicated experiment. Proceedings of the 10th International Symposium on Software Metrics, Sep. 11-17, IEEE Xplore Press, Chicago, pp: 269-278. DOI: 10.1109/METRIC.2004.1357909

Liu, S., J. Sun, Y. Liu, Y. Zhang and B. Wadhwa *et al*., 2014. Automatic early defects detection in use case documents. Proceedings of the 29th International Conference on Automated Software Engineering, Sep. 15-19, ACM, Vasteras, Sweden, pp: 785-790. DOI: 10.1145/2642937.2642969

Lopes, A., A.B. Marques, T. Conte and S.D.J. Barbosa, 2015. MoLVERIC: An inspection technique for MoLIC diagrams. Proceedings of the International Conference: Software Engineering and Knowledge Engineering, (EKE'15), pp: 13-17. DOI: 10.18293/SEKE2015-069

Ma, C., J.R. Li, C.L. Sun, J.D. Yang and Z.F. Zhou, 2014. A defects classification method for aerospace measurement and control software. Proceedings of the International Conference on Control Engineering and Information Systems, CRC Press, pp: 99-103.

Mäntylä, M.V. and J. Itkonen, 2014. How are software defects found? The role of implicit defect detection, individual responsibility, documents and knowledge. Inform. Software Technol., 56: 1597-1612. DOI: 10.1016/j.infsof.2013.12.005

Mello, R.M., E.N. Teixeira, M. Schots, C.M.L. Werner and G.H. Travassos, 2012. Checklist-based inspection technique for feature models review. Proceedings of the Brazilian Symposium on Software Components Architectures and Reuse, Sep 23-28, IEEE Xplore Press, Natal, Brazil, pp: 140-149. DOI: 10.1109/SBCARS.2012.25

Mello, R.M., W.M. Pereira and G.H. Travassos, 2010. Activity diagram inspection on requirements specification. Proceedings of the Brazilian Symposium on Software Engineering, Sep. 27- Oct. 1, IEEE Xplore Press, Salvador, pp: 168-177. DOI 10.1109/SBES.2010.29

Mendeley, 2014. Mendeley desktop v1.11@inproceedings.

Miller, J. and Z. Yin, 2003. Adding diversity to software inspections. Proceedings of the 2nd International Conference on Cognitive Informatics, Aug. 20-20, IEEE Xplore Press, London, pp: 81-86. DOI: 10.1109/COGINF.2003.1225957

Miller, J., M. Wood and M. Roper, 1998. Further experiences with scenarios and checklists. Empirical Software Eng., 3: 37-64. DOI: 10.1023/A:1009735805377

Mishra, D. and A. Mishra, 2009. Simplified software inspection process in compliance with international standards. Comput. Standards Interfaces, 31: 763-771. DOI: 10.1016/j.csi.2008.09.018

Mohabbati. B., M. Asadi, D. Gašević, M. Hatala and H.A. Müller, 2013. Combining service-orientation and software product line engineering: A systematic mapping study. Inform. Software Technol., 55: 1845-1859. DOI: 10.1016/j.infsof.2013.05.006

Mohammed, A.O., Z.A. Abdelnabi, A.M. Maatuk and A.S. Abdalla, 2015. An experimental study on detecting semantic defects in object-oriented programs using software reading techniques. Proceedings of the International Conference on Engineering and MIS, Sep. 24-26, ACM Press, Istanbul, Turkey, pp: 1-6. DOI: 10.1145/2832987.2833025

Munson, J.C., A.P. Nikora and J.S. Sherif, 2006. Software faults: A quantifiable definition. Adv. Eng. Software, 37: 327-333. DOI: 10.1016/j.advengsoft.2005.07.003

Naveed, A. and N. Ikram, 2015. A novel checklist: comparison of CBR and PBR to inspect use case specification. Commun. Comput. Inform. Sci., 558: 109-125.

Neto, P.A.M.S., I.C. Machado, J.D. McGregor, E.S. Almeida and S.R.L. Meira, 2011. A systematic mapping study of software product lines testing. Inform. Software Technol., 53: 407-423. DOI: 10.1016/j.infsof.2010.12.003

Novais, R.L, A. Torres, T.S. Mendes, M. Mendonc and N. Zazworka, 2013. Software evolution visualization: A systematic mapping study. Informa. Software Technol., 55: 1860-1883. DOI: 10.1016/j.infsof.2013.05.008

Petersen, K., K. Rönkkö and C. Wohlin, 2008b. The impact of time controlled reading on software inspection effectiveness and efficiency: A controlled experiment. Proceedings of the 2nd International Symposium on Empirical Software Engineering and Measurement, Oct. 09-10, ACM, Kaiserslautern, Germany, pp: 139-148. DOI: 10.1145/1414004.1414029

Petersen, K., R. Feldt, S. Mujtaba and M. Mattsson, 2008a. Systematic mapping studies in software engineering. 12th International Conference on Evaluation and Assessment in Software Engineering, Jun. 26-27, British Computer Society, University of Bari, Italy, pp: 68-77.

Porter, A., H. Siy, A. Mockus and L. Votta, 1998. Understanding the sources of variation in software inspections. ACM Trans. Software Eng. Methodol., 7: 41-79. DOI: 10.1145/268411.268421

Pressman, R.S., 2014. Software Engineering: A Practitioner's Approach. 8th Edn., McGraw-Hill, ISBN-10: 0078022126, pp: 976.

Rawal, B.S. and A.K. Tsetse, 2016. Analysis of bugs in Google security research project database. Proceedings of the Recent Advances in Intelligent Computational Systems, Dec. 10-12, IEEE Xplore Press, Trivandrum, India, pp: 116-121. DOI: 10.1109/RAICS.2015.7488399

Rocha, A.C., F. Ramalho and P.D.L. Machado, 2015. Automating test-based inspection of design models. Software Quality J., 23: 3-28. DOI: 10.1007/s11219-013-9219-x

Rodriguez, D., I. Herraiz, R. Harrison, J. Dolado and J.C. Riquelme, 2014. Preliminary comparison of techniques for dealing with imbalance in software defect prediction. Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering, May 13-14, ACM, London, England, pp: 1-10. DOI: 10.1145/2601248.2601294

Roper, M., M. Wood and J. Miller, 1997. An empirical evaluation of defect detection techniques. Inform. Software Technol., 39: 763-775. DOI: 10.1016/S0950-5849(97)00028-1

Sabaliauskaite, G., F. Matsukawa, S. Kusumoto and K. Inoue, 2002a. An experimental comparison of checklist-based reading and perspective-based reading for UML design document inspection. Proceedings of the International Symposium on Empirical Software Engineering, Oct. 3-4, IEEE Xplore Press, Nara, Japan, pp: 148-157. DOI: 10.1109/ISESE.2002.1166934

Sabaliauskaite, G., F. Matsukawa, S. Kusumoto and K. Inoue, 2002b. An experimental comparison of checklist-based reading and perspective-based reading for UML design document inspection. Proceedings of the International Symposium on Empirical Software Engineering (ISESE), Nara, Japan, pp: 148-157. DOI: 10.1109/ISESE.2002.1166934

Sabaliauskaite, G., F. Matsukawa, S. Kusumoto and K. Inoue, 2003. Further investigations of reading techniques for object-oriented design inspection. Inform. Software Technol., 45: 571-585. DOI: 10.1016/S0950-5849(03)00044-2

Sabaliauskaite, G., S. Kusumoto and K. Inoue, 2004. Assessing defect detection performance of interacting teams in object-oriented design inspection. Inform. Software Technol., 46: 875-886. DOI: 10.1016/j.infsof.2004.03.004

Saito, S., M. Takeuchi, S. Yamada and M. Aoyama, 2014. RISDM: A requirements inspection systems design methodology: Perspective-based design of the pragmatic quality model and question set to SRS. Proceedings of the 22nd International Requirements Engineering Conference, Aug. 25-29, IEEE Xplore Press, Karlskrona, Sweden, pp: 223-232. DOI: 10.1109/RE.2014.6912264

Sauer, C., D.R. Jeffery, L. Land and P. Yetton, 2000. The effectiveness of software development technical reviews: A behaviorally motivated program of research. IEEE Trans. Software Eng. 26: 1-14. DOI 10.1109/32.825763

Silva, N. and M. Vieira, 2016. Software for embedded systems: a quality assessment based on improved odc taxonomy. Proceedings of the 31st Annual Symposium on Applied Computing, Apr. 04-08, ACM, Pisa, Italy, pp: 1780-1783. DOI: 10.1145/2851613.2851908

Silva, N., J.C. Cunha and M. Vieira, 2016. A field study on root cause analysis of defects in space software. Reliability Eng. Syst. Safety, 158: 213-229. DOI: 10.1016/j.ress.2016.08.016

Simidchieva, B.I., L.A. Clarke and L.J. Osterweil, 2007. Representing process variation with a process family. Proceedings of the International Conference on Software Process, May 19-20, Springer-Verlag, Minneapolis, MN, USA, pp: 109-120. DOI: 10.1007/978-3-540-72426-1_10

Singh, D.A.A.G., A.E. Fernando and E.J. Leavline, 2016. Experimental study on feature selection methods for software fault detection. Proceedings of the International Conference on Circuit, Power and Computing Technologies, Mar. 18-19, IEEE Xplore Press, Nagercoil, India, pp: 1-6. DOI: 10.1109/ICCPCT.2016.7530156

Sommerville, I., 2015. Software Engineering. 10th Edn., ADDISON WESLEY Publishing Company Incorporated, Boston, ISBN-10: 0133943038, pp: 816.

Souza, I.S., G.S.S. Gomes, P.A.M.S. Neto, I.C. Machado and E.S. Almeida *et al*., 2013. Evidence of software inspection on feature specification for software product lines. J. Syst. Software, 86: 1172-1190. DOI: 10.1016/j.jss.2012.11.044

Staron, M., L. Kuzniarz and C. Thurn, 2005. An empirical assessment of using stereotypes to improve reading techniques in software inspections. Proceedings of the 3rd Workshop on Software Quality, May 17-17, ACM, St. Louis, Missouri, pp: 1-7. DOI: 10.1145/1083292.1083308

Tang, H., T. Lan, D. Hao and L. Zhang, 2015. Enhancing defect prediction with static defect analysis. Proceedings of the 7th Asia-Pacific Symposium on Internetware, Nov. 06-06, ACM, Wuhan, China, pp: 43-51. DOI: 10.1145/2875913.2875922

Teixeira, E.N., R.M. De Mello, R.C. Motta, C.L.M. Werner and A. Vasconcelos, 2015. Verification of software process line models: A checklist-based inspection approach. Proceedings of the 17th Iberoamerican Conference on Software Engineering, (CibSE), At Lima, Peru, pp: 81-91.

Thelin, T., Runeson and P. Wohlin, 2003. Prioritized use cases as a vehicle for software inspections. IEEE Software, 20: 30-33. DOI 10.1109/MS.2003.1207451

Tørner, F., M. Ivarsson, F. Pettersson and P. Öhman, 2006. Defects in automotive use cases. Proceedings of the International Symposium on Empirical Software Engineering, Sep. 21-22, ACM, Rio de Janeiro, Brazil, pp: 115-123. DOI: 10.1145/1159733.1159753

Travassos, G.H., 2014. Software defects: Stay away from them. Do inspections!. Proceedings of the 9th International Conference on the Quality of Information and Communications Technology, Sep. 23-26, IEEE Xplore Press, Guimaraes, pp: 1-7. DOI: 10.1109/QUATIC.2014.8

Travassos, G.H., F. Shull and J. Carver, 2001. Working with UML: A software design process based on inspections for the unified modeling language. Adv. Comput., 54: 35-98. DOI: 10.1016/S0065-2458(01)80015-2

Travassos, G.H., F.J. Shull, M. Fredericks and V.R. Basili, 1999. Detecting defects in object-oriented designs: Using reading techniques to increase software quality. Proceedings of the 14th Conference on Object-Oriented Programming, Systems, Languages and Applications, Nov. 01-05, ACM Press, Denver, Colorado, pp: 47-56. DOI: 10.1145/320384.320389

Valentim, N.M.C., J. Rabelo, A.C. Oran, T. Conte and S. Marczak, 2015. A controlled experiment with usability inspection techniques applied to use case specifications: Comparing the MIT 1 and the UCE techniques. Proceedings of the 18th International Conference on Model Driven Engineering Languages and Systems, Sep. 30-Oct. 2, IEEE Xplore Press, Ottawa, pp: 206-215. DOI: 10.1109/MODELS.2015.7338251

van Lamsweerde, A., 2009. Requirements Engineering: From System Goals to UML Models to Software Specifications. 1st Edn., Wiley, Hoboken, N.J., ISBN-10: 0470012706, pp: 712.

Wagner, S., 2006. A model and sensitivity analysis of the quality economics of defect-detection techniques. Proceedings of the International Symposium on Software Testing and Analysis, Jul. 17-20, ACM, Portland, Maine, pp: 73-84. DOI: 10.1145/1146238.1146247

Walia, G.S. and J.C. Carver, 2009. A systematic literature review to identify and classify software requirement errors. Inform. Software Technol., 51: 1087-1109. DOI: 10.1016/j.infsof.2009.01.004

Wieringa, R., N. Maiden, N. Mead and C. Rolland, 2005. Requirements engineering paper classification and evaluation criteria: A proposal and a discussion. Requirements Eng., 11: 102-107. DOI: 10.1007/s00766-005-0021-6

Winkler, D. and S. Biffl, 2015. Focused inspections to support defect detection in automation systems engineering environments. Proceedings of the International Conference on Product-Focused Software Process Improvement, (SPI'15), Springer, pp: 372-379. DOI: 10.1007/978-3-319-26844-6_27

Winkler, D., B. Thurnher and S. Biffl, 2007. Early software product improvement with sequential inspection sessions: An empirical investigation of inspector capability and learning effects. Proceedings of the 33rd EUROMICRO Conference on Software Engineering and Advanced Applications, Aug. 28-31, IEEE Xplore Press, Lubeck, Germany, pp: 245-254. DOI: 10.1109/EUROMICRO.2007.28

Wu, J., S. Lv, E. Ding and B. Luo, 2015. A case study in specification defects detection using statecharts. Proceedings of the 6th International Conference on Software Engineering and Service Science, Sep. 23-25, IEEE Xplore Press, Beijing, China, pp: 1-6. DOI: 10.1109/ICSESS.2015.7338994

Yousef, A.H., 2014. Extracting software static defect models using data mining. Ain Shams Eng. J., 6: 133-144. DOI: 10.1016/j.asej.2014.09.007

Yusop, N.S.M., J. Grundy and R. Vasa, 2016. Reporting usability defects: Do reporters report what software developers need? Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering, Jun. 01-03, ACM, Limerick, Ireland, pp: 1-10. DOI: 10.1145/2915970.2915995

Zhu, Y.M., 2016. Software reading techniques: Twenty techniques for more effective software review and inspection. 1st Edn., Apress, Berkeley, CA, ISBN-10: 978-1-4842-2345-1, pp: 126.