Experimental Studies in Software Inspection Process
A Systematic Mapping

Elis Montoro Hernandes[1], Anderson Belgamo[2] and Sandra Fabbri[1]
[1] *LaPES - Software Engineering Research Lab, Federal University of São Carlos, UFSCar, São Carlos, SP, Brazil*
[2] *IFSP - São Paulo Federal Institute of Education, Science and Technology, Piracicaba, SP, Brazil*

Keywords: Systematic Mapping, Software Inspection Process, Experiments, Empirical Software Engineering, Experimental Software Engineering.

Abstract: *Background*: The interest in produce experimental knowledge about verification and validation techniques increases over the years. This kind of knowledge can be useful for researchers who develop studies in that area as well as for industry that can make decisions about verification and validation activities (V&V) on the basis of experimental results. *Aim*: This paper aims to map the empirical studies conducted in the software inspection process area. *Method*: Each step of the Systematic Mapping (SM) process was performed with the support of the StArt tool, and the papers from major databases, journals, conferences, and workshops were covered. *Results*: Seventy nine papers were accepted in this mapping and helped identifying the inspection processes, techniques and tools commonly referenced in that papers, as well as the artifacts usually inspected and the research groups and universities frequently involved in these papers. *Conclusion*: Different inspection processes have been investigated through experimental studies, and the Fagan's process is the most investigated of them. To evaluate these different processes requirements document and source code were the artifacts more used. Besides, different tools and techniques have been used to support these processes. Some important lessons were learned, which are in accordance to explanations of others authors.

1 INTRODUCTION

The inspection activity is a systematic approach that aims to detect defects in software artifacts as soon as they are committed. Since it was introduced in IBM around 70's, by Michael Fagan, the inspection activity is considered one of the best software engineering practices to identify defects (Anderson et al., 2003).

One of the inspection activity advantages is that it can be applied on many kinds of software artifacts, as soon as these are constructed, decreasing defects transfer to other artifacts. besides the possibility of being applied them before the testing activities (Boogerd and Moonen, 2006).

The software inspection process proposed by Fagan (1976) has been modified and adapted, generating other versions such as Fagan (1986), Humphrey (1989), NASA (1993), Gilb and Graham (1993), Murphy and Miller (1997), Sauer et al., (2000), Halling et al., (2003), and Denger and Elberzhager (2007).

Although each different version of the software inspection process has proposed new and different roles and activities, all of them share the same goal: allow inspectors identifying defects, analyzing them and establishing the real defects for correction.

When an enterprise decides to adopt an inspection process, it is necessary to identify which process and which techniques fit better for its development process and team.

A way to search for evidence about what process or technique to use, is to look for experimental studies related to them. According to Travassos et al., (2002), software engineering experimental studies aim to characterize, evaluate, foresee, control and improve products, process, resources, models and theories. In addition, Basili et al., (1996) argue that "*the only way to discover how applicable a new method, technique, or tool is in a given environment is to experiment with its use in that environment*".

Thus, the importance of experimental studies is clear for software engineering researchers and enterprises that want to adopt or adapt processes and techniques in their business environment. Well-known techniques to find this kind of knowledge are

Systematic Review (SR) (a.k.a. Systematic Literature Review (SLR)) and Systematic Mapping (SM).

Although the research process is benefited by these techniques, the SR process is considered laborious to be planned and conducted (Kitchenham, 2004). It demands a high maturity level from the researcher to define the right research question to be answered, leading novice researchers to opt for an ad-hoc literature review.

According to Bailey et al., (2007), Systematic Mappings (SM), also known as Scoping Review (Petticrew and Roberts, 2006), should be conducted before a SR, since its goal is to identify the features and the kind of publications to be investigated on a particular theme.

The main goal of Systematic Mapping studies is to provide an overview of a research area, and identify the quantity and type of research and results available within it as well as mapping the frequencies of publication over time to see trends." (Petersen et al., 2008). These authors also mentioned that a secondary goal is to identify the forums where the research topic has been published. Furthermore, Systematic Mapping studies can help in refining the question for the full review and estimating the resources that will be needed (Petticrew and Roberts, 2006).

The goal of this paper is to describe a Systematic Mapping of experimental studies related to software inspection process. The objective was to identify techniques, tools, artifacts and the inspection process usually employed in the published experimental studies. Moreover, the intention was to identify the process and techniques more explored in the experimental studies, their respective authors and the places where the studies are usually published.

The remaining of this paper is organized as follows: Section 2 presents the methodology applied to conduct this SM; Section 3 presents the results; Section 4 presents the threats of validity; and finally, Section 5 presents the discussions regarding the study.

2 METHODOLOGY

Briefly, SM allows creating an overview of an interest area based on the definition of research questions and the identification and quantification of the collected data.

Petersen et al., (2008) present the following steps as the essential process steps for a Systematic Mapping study (Figure 1):

1) *Definition of research question:* In this step, one or more research questions should be defined, reflecting the expected answer at the end of the mapping. The outcome of this step is the review scope.
2) *Conduct Search*: In this step, search strings are defined based on the research questions established in the previous step. The search strings are then applied to different online scientific databases to identify relevant papers for the mapping. The outcome of this step is a list of papers retrieved by the search strings.
3) *Screening of Papers*: In this step, inclusion and exclusion criteria should be applied based on the research questions. These criteria are intended to identify those primary studies that provide direct evidence about the research question. In order to reduce the likelihood of bias, selection criteria should be established during the protocol definition, although they may be refined during the search process (Kitchenham, 2007). The outcome of this step is a list of relevant papers to the research theme.
4) *Keywording using Abstracts*: In this step, the researcher must read the abstract of accepted papers and identify keywords that characterize various aspects of the studies, like the research method, type of conducted study, research area, research group, method and/or tool used, etc. After the reading, a set of keywords is created and used to classify all papers in different features. The outcome of this step is a Classification Scheme.
5) *Data Extraction and Mapping Process*: the accepted papers in step 3 are classified according to the categories previously identified in step 4. The classification scheme may evolve during the data extraction, either by adding new categories or merging or splitting existing categories. After that, the categories are grouped into facets, which in turn are related between each other to generate a map (as a bubble plot, for example) so to allow the researcher to visualize various aspects of the studied research topic. The outcome of this step is the generated map.

Next sections present the goal of the SM process steps and how they were conducted.

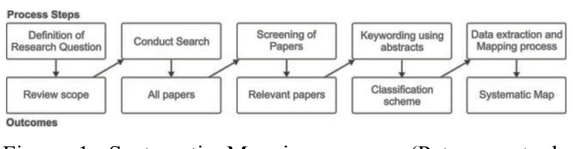

Figure 1: Systematic Mapping process (Petersen et al., 2008).

2.1 Definition of Research Question

According to the intention of this SM, presented in Section 1, the research questions were:

RQ.1)"Which software inspection processes were investigated through experimental studies?".

RQ.2)"Which techniques and tools have been used in experimental studies that investigated software inspection processes?".

Petersen et al., (2008) suggest that a protocol is filled as it is in Systematic Reviews to enable the registration of the study decisions, as well as the auditing and replication.

Aiming to use a computational support for conducting this SM, the StArt tool (Hernandes et al., 2010); (Zamboni at el, 2010); (Fabbri et al., 2012) was used. Hence, the protocol was based on Kitchenham's proposal (2007), since this is the model provided by the tool. Although the protocol data can be adjusted during the process execution, in this step the research questions, the inclusion and exclusion studies criteria and the information extraction form fields were defined.

2.2 Conduct Search

The definition of the search string is relevant to ensure that the studies to be analyzed support the answer to the research question. The online scientific database SCOPUS was selected to perform essays till an effective search string was reached.

SCOPUS was chosen because it offers facilities that allow operations with a set of strings besides analyzing relevant data as: list of research area, authors, conferences/journals and keywords most frequent in the papers retrieved. Furthermore, Dieste et al., (2009) argue that it has fewer weaknesses than the other online scientific databases.

After some essays, the search string defined was:

TITLE-ABS-KEY("software inspection process" OR (("inspection process" OR "inspection technique") AND "software engineering")) AND TITLE-ABS-KEY("primary stud" OR "experiment*" OR "empirical stud*" OR "controlled stud*")*

After performing the query in SCOPUS and exporting the results (papers) to a .bib file, the same search string was adapted to be applied to the other online scientific database defined in the protocol: IEEExplore, ACM Digital Library and Web of Science.

These four online scientific databases were inserted into the StArt tool protocol such that a respective search session was created for each one. Aiming to enable replications of this study, for each search session the respective search string was registered in the tool, as well as the BibTex file was imported.

Dieste et al., (2009) exposed that the results of an ACM DL search cannot be exported either to text or Reference Manager formats which represents a significant impediment for a SR.

To workaround this problem we used the Zotero Firefox plug-in (*www.zotero.org*) to import the ACM DL results and generate the BibTex file.

When the user imports a *.bib* file to a search session of StArt, duplicated papers (with the same title and authors) are automatically identified. In addition, the tool sets a score for each papers, which is based on the frequency that each keyword (defined in the protocol) appears in the title, abstract and keywords of the paper.

After all .bib files were imported, 249 papers were inserted under this SM, having 116 as duplicated papers (indexed by more than one online scientific database).

2.3 Screening of Papers

The computational support of the StArt tool makes the screening of papers easier. The tool offers an interface that allows the reading of the abstracts (retrieved by .bib reference file); shows the inclusion and exclusion criteria (defined on the protocol); enables the attribution of these criteria to the papers; and allows setting the papers as accepted or rejected. The StArt also allows the user setting a reading priority (very high, high, low and very low), deduced from the reading of the abstract, which will be useful when the full paper should be read, for example, in a Systematic Review.

In order to revise the inclusion and exclusion criteria and the data extraction form (both defined in the protocol - Step 1), first of all the five high scored and five low scored papers were analyzed.

The criteria were satisfactory and no change was made. By the other hand, a new field was added to the data extraction form: "*Which process phase (or activity) is supported by the technique?*" (more details in Section 2.5).

Although Petersen et al., (2008) suggest that this SM step is conducted based on the paper abstract (or some paper sections), there are cases where the abstract is not enough for applying the inclusion and exclusion criteria. In these cases, reading the full text can be a good option, like in the Systematic Review process (Kitchenham, 2004; 2007).

At the end of this step, 54 papers were rejected and 79 were accepted. Table 1 shows the inclusion and exclusion criteria applied and the number of papers related to each one. There are some papers that were related to more than one criterion.

It is important to mention that this systematic mapping has focused just in papers about software inspection process and its variants – papers about peer review, as example, were not considered.

Table 1: Inclusion and Exclusion criteria applied in the SM.

Criterion type	Criterion	Number of papers
Inclusion	presents or uses some tool or technique to support the experimental studies in software inspection process	31
	presents a experimental study related to software inspection process	56
Exclusion	proceedings introduction	1
	not written in English or Portuguese	1
	full paper not available on web neither in the university commutation service	4
	not presents an inspection process related to software	14
	not related to software	16
	not presents an experimental study related to software inspection process	18

2.4 Keywording of Abstracts

Although Petersen et al., (2008) suggest that the classification schema should emerge while the step *Keywording of Abstracts* is being conducted, in this SM it was defined when the protocol was filled. In this way, we tried to ensure that the required data to answer the research question would be extracted.

In this SM the *Keywording of Abstract* was conducted in parallel to the *Screening of Papers*. Hence, if a paper was accepted, as the abstract was read, the Classification Scheme (data extraction form) defined in the protocol was revised aiming to: i) identify if the paper would fill all items of the schema; ii) identify values to compose a list of possible categories for each classification schema item.

Again, the StArt tool makes this task easier, once it gives two possibilities to define the classification schema items in the data extraction form (resource available in the tool): textual classification or itemized classification.

If the user set a classification item as textual, this field will be a text and probably, different for each papers. If the user set a classification item as itemized, a list of categories must be created and only one category can be chosen when the item is filled. Of course, the list of categories can be updated if necessary.

The itemized item is a good option to ensure standardization of answers. For instance, in this SM the classification item "*study type*" was an itemized item and its categories were updated as the *Keywording of abstracts* was conducting.

2.5 Data Extraction and Mapping of the Studies

In this step the 79 accepted papers were categorized in 8 classification items:

a) study classification, according to Wieringa (2006);
b) inspection process used,
c) artifact inspected;
d) tool used in the study;
e) step (s) of the process supported by the tool;
f) techniques used in the study;
g) step (s) of the process supported by the technique;
h) research group or university.

In addition to the map based on data collected through the classification schema, using the StArt tool it is possible to map the research area taking into account other data, for example: publication year, conferences or journals, and authors. These fields are available when the .bib file is imported to the tool.

Considering the research question, the bubble chart that maps the research allows identifying the techniques used in experimental studies related to software inspection process and which activities are supported by these techniques.

Charts and tables show the publications evolution, processes and tools more used, artifacts commonly inspected, and so on.

When the StArt tool is used, besides the characterization of the papers by means of the Classification Schema, the papers can be characterize by additional and relevant information. This is possible since relevant information can also be registered in a memo field provided by the tool, for each paper. We emphasize that any scientific methods to Thematic Synthesis as mentioned by Cruzes and Dyba (2011) were considered.

As secondary studies should be accessible to the community (Kitchenham, 2007), Pai et al., (2004), etc) the package of this study is available at: www.dc.ufscar.br/~lapes/packs/inspecao5.1.

3 RESULTS

The results will be commented according to the research questions.

The first question is related to the software inspection process that have been used in experimental studies: *RQ.1)"Which software inspection processes were investigated through experimental studies?".*

According to Figure 2, that shows the identified processes and how many papers cited them, we observe that few authors mentioned the process used in the experimental study (23 occurrences).

The Fagan process (Fagan, 1976; 1986) was the most mentioned (22 occurrences), mainly if some adaptations of this process are also considered (7 occurrences) (Porter et al., 1995); (Porter et al., 1998); (Kelly and Shepard, 2000); (Harjumaa, 2003); (Vitharana, Ramamurthy, 2003); (Torner et al., 2006); (Porto et al., 2009).

The process presented by Sauer et.al (2000) was also highlighted among the accepted papers (7 occurrences). Some adaptations of this process (Bernardez et al., 2004); (Winkler et al., 2005); (Walia and Carver, 2008) were also mentioned (3 occurrences).

The inspection process presented by Gilb and Graham (1993) (3 occurrences), HyperCode (Perpich et al., 1997); (Perry et al., 2002) (2 occurrences) and N-fold (2 occurrences) (Schneider et al, 1992; He, Carver, 2006) were the other processes also explicitly cited among the accepted papers.

As mentioned before, although the secondary study that was carried out was a SM, which does not require the full reading of papers, some of them were completely read aiming to extract more detailed information.

As mentioned before, there were cases where a brief reading of the full text was needed. Although different processes were identified among the experimental studies, all of them propose activities to plan the inspection, find defects, analyze the defects and select the ones for rework. The main differences between the processes stayed on the roles, intermediate activities, strategies to collect and analyze defects and tools to support them.

It is important to notice that some papers used more than one inspection process. In addition, a process was considered "adapted" when some of its steps were not performed.

As software inspection can be applied to all kind of software artifact, the artifacts referred in the experimental studies were of different types. Figure 3 shows this information and highlights that the most investigated artifacts are requirement documents and code. The reviewers have decided maintain the name of the artifacts mentioned by each author, which led to present UML diagrams, Use Cases, Use Cases model and OO diagrams as different items.

The secondary question is related to techniques and tools that have been used in experimental studies: *RQ.2)"Which techniques and tools have been used in experimental studies that investigated software inspection processes?"*

Figure 4 shows a bubble chart that shows the most mentioned techniques and the respective activities supported by them. Table 2 shows the tools and what features each one provides. Only 25 papers mentioned some tool and which one was used.

Related to the techniques, as expected, most of them are reading techniques or techniques defined for other purpose but used to find defects (e.g. heuristic evaluation (Frøkjær and Hornbæk, 2008)).

The outstanding reading techniques of this SM were Checklist and Perspective-Based Reading (PBR). PBR is a systematic technique for defect detection in requirement documents and Checklist is a reading technique that can be applied for reading different types of artifacts. Notice that this result is aligned with the one showed in Figure 3, since the requirements document was mentioned as the most inspected artifact in the experimental studies.

Other technique commonly used in the experimental studies was Capture-Recapture (Runeson, Wohlin, 1998); (Miller, 1999); (Freimut et al., 2001, Thelin, 2003); (Thelin, 2004); (Walia et al., 2008). This technique is related to statistic methods used to quantify remaining defects in the artifact after the inspection. The use of Capture-Recapture in software inspection was proposed by Eick et al., (1992) and some studies and improvements were presented thereafter.

Related to the tools mentioned in the papers, they usually provide support to the software inspection process in different ways.

Regarding inspection meeting or defects discrimination activity, in general, the tools adopt asynchronous communication. The roles involved in these activities communicate and share opinions by means of forums created to enable discussions about the defects identified during the inspection, for example. A score previously assigned to each defect by inspectors assists the discussion and also helps the moderator in defining if a defect is a false-positive or a real-defect (Lanubile et al., 2004); (Kalinowski and Travassos, 2004); (Ardito et al.,

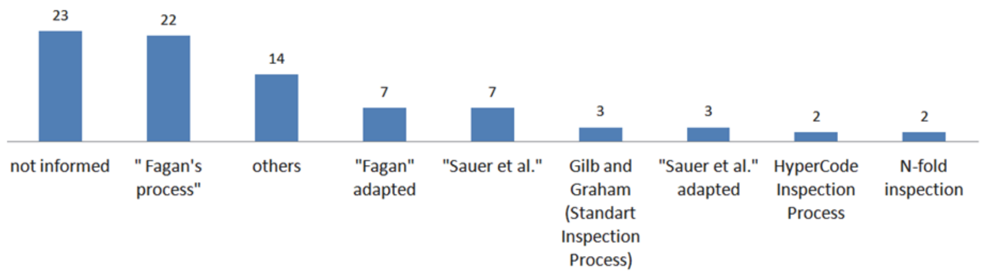

Figure 2: Software inspection process used in the experimental studies.

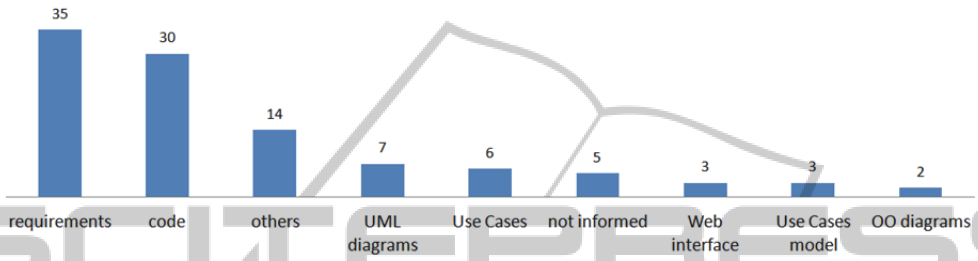

Figure 3: Artifacts inspected in the experimental studies.

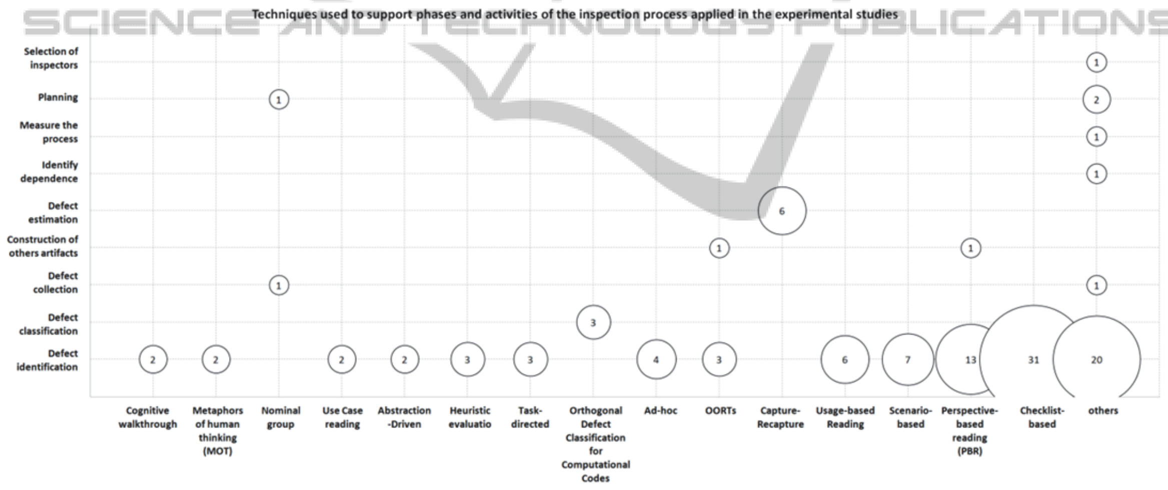

Figure 4: Techniques x Inspection Process phase/activity map.

2006b).

Two studies applied synchronous communication to support these activities. Tyran and George (2002) have used a GSS tool (Group Support System) to allow that a group of inspectors discuss the process outcomes in a synchronous way. A similar study was presented by Vitharana and Ramamurthy (2003), who used a GSS tool to enable the roles involved in the inspection joining in the discussion activity anonymously.

Although both studies presented appropriate results using this kind of tool, these tools do not assist other inspection activities such as defects identification.

Porto et al., (2009) presented CRISTA (Code Reading with Stepwise Abstraction) that supports the inspection meeting and defect discrimination activity. Despite the discussions can be performed by groups, they must be coordinated by the moderator who inserts the decision into the tool. CRISTA also supports defects detection.

As mentioned before, for SM it is also important to identify the sources where the papers were published. Hence, Table 3 shows the conferences and journals that published the accepted papers - those which published only one paper were grouped in the row "Other conferences and journals". The outstanding in this classification was the Empirical Software Engineering Journal.

Table 2: Tools used in the experimental studies.

Software inspection activities (Sauer et al., 2000) →	Planning			Discovering			Collection		Discrimination						Rework	Follow-up
Activities mentioned by the authors →	Planning	Artifacts / documents sharing	Individual preparing	Defect identification	Automatic defect identification	Defect identification form	Defects collection	Analysis and defect priorization	Defects discrimination	Data analysis	Meeting (asynchronous)	Meeting (synchronous)	Inspection review	Simulations of the process	Rework	Capture-recapture techniques application
Adobe Acrobat 5.0		✓									✓					
ASSIST-Asynchronous/Synchronous Software Inspection Support Tool	✓		✓				✓									
Assistent to Usability Inspection Process	✓			✓		✓	✓	✓	✓							
Capture-Recapture; CARE																✓
CRISTA				✓		✓			✓	✓						
Document Quality Defect Detection tool				✓		✓										
Extend Software Modeling tool														✓		
FindBugs; Jlint				✓												
Electronic form						✓										
Gerdame / NosePrints					✓					✓						
GRIP	✓					✓	✓									
GroupSystems	✓					✓	✓									
HyperCode																
InspectA						✓	✓		✓				✓			
Internet-Based Inspection System (IBIS)	✓			✓		✓	✓		✓							
ISPIS	✓			✓		✓	✓		✓						✓	
Spreadsheet						✓										
SUIT (Systematic Usability Inspection Tool)	✓			✓			✓					✓				
Ventana Corp.'s Group Outliner tool						✓						✓				
VisionQuest; GSS anonymous software						✓	✓					✓				
WAIT (web-based artefact inspection tool)	✓			✓		✓	✓		✓							
Web-based Inspection Process Support tool (WIPS)						✓	✓									

Table 4 shows the universities and research groups that have conducted experimental studies related to software inspection. The outstanding in this classification were the Vienna University of Technology and the University of Maryland. Figure 5 shows a cloud of authors' name who published the accepted papers.

Figure 6 shows the type of studies according to the categories presented by Wieringa (2006). Considering that this SM is about experimental studies that investigated software inspection processes, as expected, most of the papers corresponded to a validation or evaluation of some process. The "*proposal*" category represents the studies that presented a new proposal and also an empirical study to evidence the proposal advantages.

Table 3: List of journals/conferences which published experimental studies on software inspection.

Journals	Number of papers
Empirical Software Engineering	7
IEEE Transactions on Software Engineering	5
ACM Transactions on Software Engineering and Methodology (TOSEM)	2
Information and Software Technology	2
Lecture Notes in Computer Science	2
SIGSOFT Software Engineering Notes	2
Software Testing Verification and Reliability	2
Conferences	
International Conference on Software Engineering (ICSE)	5
International Software Metrics Symposium	4
ACM/IEEE International Symposium on Empirical Software Engineering	3
Conference of the Centre for Advanced Studies on Collaborative research	3
International Symposium on Empirical Software Engineering (ISESE)	3
Asia-Pacific Software Engineering Conference	2
Euromicro Conference	2
EUROMICRO Conference on Software Engineering and Advanced Applications, SEAA	2
International Conference on Automated Software Engineering (ASE)	2
International Conference on Quality Software (QSIC)	2
International Conference on Software Engineering and Knowledge Engineering, SEKE	2
Nordic Conference on Human-computer Interaction	2
Brazilian Symposium on Software Engineering	2
Other conference and journals	23

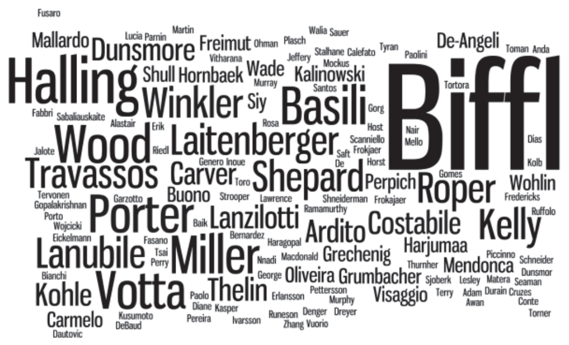

Figure 5: Authors highlighted in the accepted papers.

Table 4: Universities and research groups highlighted on the SM.

University/Research group	Number of papers
Vienna University of Technology	13
University of Maryland	8
Fraunhofer Kaiserslautern	6
University of Strathclyde	7
Federal University of Rio de Janeiro	4
Johannes Kepler University Linz	4
Lund University	4
Royal Military College of Canada	4
Università di Bari	4
AT&T Bell Labs	4
Mississippi State University	3
University of Bari	3
Federal University of São Carlos	2
Fraunhofer Maryland	2
Technical University Vienna	2
University of Copenhagen	2
University of Oulu	2
University of Sannio	2
Other universities / research groups	40

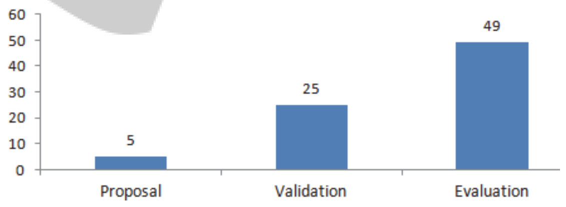

Figure 6: Studies classification according to Wieringa's categories (2006).

Figure 7 shows the distribution of accepted papers in the years. It is known that some online scientific database do not index conference papers straightway the conference closing. Taking into account the search strings was applied in the first semester of 2012 (April/May), it may explain the absence of papers published in 2012.

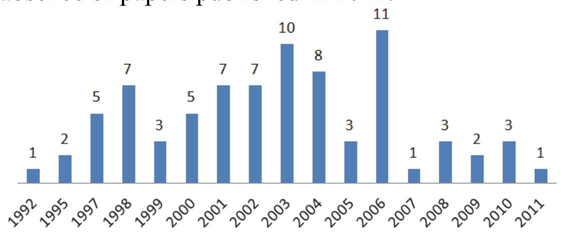

Figure 7: Distribution of publication by years.

4 THREATS TO VALIDITY

Although a research protocol have been filled and evaluated by other members of the research group, some threats to validity can be identified.

Basically, the main threats are of internal validity as: (i) researcher's bias when analyzing the primary studies; (ii) the possibilities provided by the online scientific databases of constructing the search string, which may not catch all representative papers from the database; and (iii) the researcher's university permission related to the online scientific databases access, which may cause non access to full papers and, consequently, their rejection.

Some actions were taken for minimizing these threats, for example, assessing the protocol during the screening of papers to certificate that it portrayed the correct selection criteria; and the conduction of pilot study for reaching an acceptable search string.

Regarding the details present in this paper and the study package available, we believe that a replication of this study is feasible. Even the replication date can affect the outcome and threats to validity, we believe the underlying trends should remain unchanged.

5 DISCUSSIONS, LESSONS LEARNED AND FUTURE WORKS

This paper presented a systematic mapping of experimental studies on software inspection process.

The search string was applied in four online scientific databases (SCOPUS, IEEExplore, ACM Digital Library and Web of Science) and 249 papers were retrieved. Taking title and authors into account, the StArt tool, used as computational support to conduct this study, identified 116 duplicated papers. After analyze the 133 remaining papers and apply the inclusion and exclusion criteria, 79 papers were accepted.

The data extracted from these papers showed that the inspection process presented by Fagan (1976; 1986) was the most mentioned. Although software inspection can be applied to all kind of software artifact, requirement documents and code were highlighted as the most investigated artifacts.

In relation to the techniques, the most mentioned were the reading techniques. Checklist and Perspective-Based Reading (PBR) were highlighted.

Concerning the tools, a wide list of tools with different purposes was identified. Some tools were specific to the software inspection process, such as CRISTA (Porto et al., 2009), ISPIS (Kalinowski and Travassos, 2004), HyperCode (Perry et al., 2002), InspectA (Murphy and Miller, 1997) and IBIS (Lanubile et al., 2004). Other tools were used in the experimental studies but were not for supporting the software inspection process properly, such as Capture-Recapture (Runeson and Wohlin, 1998) and FindBugs (Wojcicki and Strooper, 2006).

Some lessons learned deserve attention: the importance of good abstracts and the registration of some details about any object under evaluation through experimental studies. This is very important for reaching the objective of a SM.

Although the SM process suggests that the Classification Schema is created on the basis of papers abstract, few of them provide basic information about the conducted study. Hence, the only way to get the necessary information is to read the full paper or some section of it.

Even so, there were papers that did not exhibit relevant details about the conducted study, such as: the process used (or the way that the process was performed), artifacts inspected, how the data was analyzed and the threats to validity. Thus, the lack of information for filling the classification schema established by the investigators can jeopardize the research area characterization.

These hardships faced by the authors (lack of information both in the abstracts and the full paper) emphasize the importance of topics already explained by other authors: structured abstracts (Budgen et al., 2008) and guidelines to report empirical studies (Jedlitschka and Pfahl, 2005).

Considering that this SM was conducted in the context of a PhD research that aims to give better support to the inspection meeting and defect discrimination activities, as future work the most used inspection processes will be investigated more deeply. Hence, systematic reviews are being planned and will be conducted as future work.

ACKNOWLEDGEMENTS

Our thanks to the Brazilian funding agencies CAPES and CNPq and INEP (*Observatório da Educação*, project 3280). E. M.H thanks CAPES foundation (BEX 11833/12-2) and the Software Engineering Group of The University of Alabama.

REFERENCES

Anderson, P.; Reps, T.; Teitelbaum, T. 2003. Design and implementation of a fine-grained software inspection tool. *IEEE Transactions on Software Engineering*, 29(8), pp. 721-733.

Ardito, C., Lanzilotti, R., Buono, P., Piccinno, A. 2006. A tool to support usability inspection. In: *Working Conference on Advanced Visual Interfaces*, 11th. AVI. Veneza, Italy, May, 2006. New York: ACM Press.

Basili, V.R., Green, S., Laitenberger, O., Shull, F., Sørumgård, S., Zelkowitz, M. 1996. The empirical investigation of Perspective-Based Reading. *Empirical Software Engineering*, 1(2), pp. 133-164.

Bailey, J., Budgen, D., Turner, M., Kitchenham, B., Brereton, P., and Linkman, S. (2007). Evidence relating to Object-Oriented software design: A survey. In: *International Symposium on Empirical Software Engineering and Measurement,* Madri, Spain, Sep., 2004. Los Alamitos: IEEE Press.

Bernardez, B.; Genero, M.; Duran, A.; Toro, M. 2004. A controlled experiment for evaluating a metric-based reading technique for requirements inspection. In: International Symposium on Software Metrics, 2004. Proceedings. 10th, Chicago, USA, Sep., 2004. Los Alamitos: IEEE Press.

Boogerd, C., Moonen, L. 2006. Prioritizing Software Inspection Results using Static Profiling. In *IEEE International Workshop on Source Code Analysis and Manipulation*, 6th. SCAM. Philadelphia, USA. Dec., 2006. Los Alamitos: IEEE Computer Society.

Budgen, D.; Kitchenham, B.; Charters, S.; Turner, M.; Brereton, P.; Linkman, S. 2008. Presenting software engineering results using structured abstracts: a randomised experiment. Empirical Software Engineering. 13(4). pp. 435--468.

Cruzes, D.; Dyba, T. 2011. Recommended Steps for Thematic Synthesis in Software Engineering. In: *International Symposium on Empirical Software Engineering and Measurement*, Banff, Canada, Sep. 2011. Los Alamitos: IEEE Computer Society.

Denger, C., Elberzhager, F. 2007. Unifying inspection processes to create a framework to support static quality assurance planning. In: *Euromicro Conference on Software Engineering and Advanced Applications*, 33rd, Lubeck, Germany, Aug., 2007. Los Alamitos: IEEE Press.

Dieste, O.; Grimán, A.; Juristo, N. 2009. Developing search strategies for detecting relevant experiments. *Empirical Software Engineering*. 14(5). pp 513-539.

Eick, S., Loader, C., Long, D., Votta, L., and Vander Wiel, S. 1992. Estimating software fault content before coding. In: *International Conference on Software Engineering*, 14th, Melbourne, Australia, May., 1992. Los Alamitos: IEEE Computer Society.

Fabbri, S., Hernandes, E., Di Thommazo, A., Belgamo, A., Zamboni, A., Silva, C. 2012. Managing literature reviews information through visualization. In *International Conference on Enterprise Information Systems*.14th. ICEIS, Wroclaw, Poland, Jun, 2012. New York: SCITEPRESS.

Fagan, M. E. 1976. Design and code inspections to reduce errors in program development. *IBM Systems Journal*, 15(7), pp. 182-211.

Fagan, M. E. 1986. Advances in software inspections. *IEEE Transactions on software Engineering*, 12(7), pp. 744-751.

Freimut, B.; Laitenberger, O.; Biffl, S.. 2001. Investigating the impact of reading techniques on the accuracy of different defect content estimation techniques. In: *Software Metrics Symposium*, 17th., London, UK, Apr., 2001. Los Alamitos: IEEE Press.

Frøkjær, E; Hornbæk, K. 2008. Metaphors of Human Thinking for Usability Inspection and Design. *ACM Transactions on Computer-Human Interaction*. 14(4). pp. 1--33.

Gilb, T., Graham, D. 1993. *Software Inspection*. Wokingham, England: AddisonWesley,.

Halling, M., Biffl, S., Grunbacher, P. 2002. A groupware-supported inspection process for active inspection management. In: *Euromicro Conference*, 28th. Dortmund, Germany, Dec., 2002. Los Alamitos: IEEE Computer Society.

Harjumaa, L. 2003. Distributed software inspections - An experiment with Adobe Acrobat. In: *International Conference on Computer Science and Technology*, 26th. Cancun, Mexico, May, 2003. New York: IASTED.

He, L, Carver, J. 2006. PBR vs. checklist: a replication in the n-fold inspection context. In: ACM/IEEE International Symposium on Empirical Software Engineering, Rio de Janeiro, Brazil, Sep., 2006. New York: ACM Press.

Hernandes, E. C. M.; Zamboni, A. B.; Thommazo, A. D.; Fabbri, S. C. P. F. 2010. Avaliação da ferramenta StArt utilizando o modelo TAM e o paradigma GQM. In: *X Experimental Software Engineering Latin American Workshop*, ICMC-São Carlos.

Humphrey, W. S. 1989. *Managing the software process*. Addison-Wesley Longman Publishing Co.

Jedlitschka, A.; Pfahl, D. 2005. Reporting guidelines for controlled experiments in software engineering. In: *International Symposium on Empirical Software Engineering*, Kaiserslautern, Germany, Nov. 2005. Los Alamitos: IEEE Press.

Kalinowski, M., Travassos, G.H. 2004. A computational framework for supporting software inspections. In: *International Conference on Automated Software Engineering*, 19th, Linz, Austria, Set., 2004. Los Alamitos: IEEE Computer Society.

Kelly, D.; Shepard, T. 2000. Task-directed software inspection technique: an experiment and case study. In Conference of the Centre for Advanced Studies on Collaborative research, Mississauga, Canada, Nov., 2000. Palo Alto: IBM Press.

Kitchenham, B. A. 2004. *Procedures for Performing Systematic Reviews*. Software Engineering Group, Keele University, Keele, Tech. Rep. TR/SE 0401.

Kitchenham, B. A. 2007. *Guidelines for performing*

Systematic Literature Reviews in Software. Software Engineering Group, Keele Univ., Keele, Univ. Durham, Durham, Tech. Rep. EBSE-2007-01.

Lanubile, F.; Mallardo, T.; Calefato, F. 2004. Tool support for geographically dispersed inspection teams. *Software Process: Improvement and Practice*, 8(4), pp. 217-231.

Miller, J. 1999. Estimating the number of remaining defects after inspection. *Software Testing, Verification and Reliability*. University of Strathclyde, UK.

Murphy, P., Miller, J. 1997. A process for asynchronous software inspection. In: *IEEE International Workshop on Software Technology and Engineering Practice*, 8th, Jul., 1997, London, UK. Los Alamitos: IEEE Press.

NASA. 1993. *Software Formal Inspections Guidebook*. Washington, USA, Aug., 1993. Available at: <http://www.cs.nott.ac.uk/~cah/G53QAT/fi.pdf>. Accessed: 07/Feb/2013.

Pai, M., McCulloch, M., Gorman, J. D., Pai, N., Enanoria, W., Kennedy, G., Tharyan, P., Colford Jr., J. M. 2004. *Clinical Research Methods - Systematic reviews and meta-analyses: An illustrated, step-by-step guide.* The National Medical Journal of India.

Perpich, J.; Perry, D.; Porter, A.; Votta, L.; Wade, M. 1997. Anywhere, Anytime Code Inspections: Using the Web to Remove Inspection Bottlenecks in Large-Scale Software Development. In: *International Conference on Software Engineering*, Boston, USA, May, 1997. Los Alamitos: IEEE Computer Society.

Perry, D.; Porter, A.; Wade, M.; Votta, L.; Perpich, J. 2002. Reducing inspection interval in large-scale software development. *IEEE Transaction Software Engineering*. 28(7). pp. 695--705.

Petersen, K. et al. 2008. Systematic Mapping Studies in Software Engineering, In: *Proc. Inter. Conf. on Evaluation and Assessment in Software Engineering*, Bari, Italy.

Petticrew, M. and Roberts, H. 2006. Systematic Reviews in the Social Sciences - a practical guide. Blackwell Publishing, Malden.

Porter, A., Votta, L., Basili, V. 1995. Comparing Detection Methods for Software Requirements Inspections: A Replicated Experiment. *IEEE Transactions on Software Engineering*, 21(6), pp. 563-575.

Porter, A., Siy, H., Mockus, A., Votta, L. 1998. Understanding the Sources of Variation in Software Inspections. *ACM Transactions on Software Engineering and Methodology*, 7(1), pp. 41-79.

Porto, D., Mendonca, M., Fabbri, S. 2009. The use of reading technique and visualization for program understanding. In: International Conference on Software Engineering and Knowledge Engineering, 21st. Boston, USA, May, 2009. New York: ACM Press.

Runeson, P., Wohlin, C. 1998. An Experimental Evaluation of an Experience-Based Capture-Recapture Method in Software Code Inspections. *Empirical Software Engineering*. 3(4), pp.381-406.

Sauer, C. Jeffery, D.; Land, L.; Yetton, P. 2000. The effecticveness of software development technical review: a behaviorally motivated program of research. *IEEE Transactions on Software Engineering*, 1(26), pp. 1--14.

Schneider, M; Martin, J.; Tsai. 1992. An experimental study of fault detection in user requirements documents. ACM Transactions on Software Engineering and Methodology, 1(2). pp. 188-- 204.

Thelin, T. 2003. Empirical Evaluations of Usage-Based Reading and Fault Content Estimation for Software Inspections. *Empirical Software Engineering*, 8(3).

Thelin. T. 2004. Team-Based Fault Content Estimation in the Software Inspection Process. In International Conference on Software Engineering, 26th ICSE, St Louis, USA, May, 2004. Washington: IEEE Computer.

Tyran, C. K., George, J. F. 2002. Improving software inspections with group process support. *Communications of the ACM*. 45(9), pp. 87-92.

Torner, T,; Ivarsson, M.; Pettersson, F.; Öhman, P. 2006. Defects in automotive use cases. In: *ACM/IEEE International Symposium on Empirical Software Engineering*, Redondo Beach, USA, Aug., 2006. New York: ACM Press.

Travassos, G. H., Gurov, D., Amaral, E. 2002. *Introdução à Engenharia de Software Experimental*. UFRJ. Rio de Janeiro, Brazil. 2002. (Technical Report 590/02).

Vitharana, P., Ramamurthy, K. 2003. Computer-mediated group support, anonymity, and the software inspection process: An empirical investigation. *IEEE Transactions on Software Engineering*. 29(2). pp. 167-180.

Walia, G.; Carver, J. 2008. Evaluation of capture-recapture models for estimating the abundance of naturally-occurring defects. In: *ACM-IEEE International Symposium on Empirical software Engineering and Measurement*, 2nd, Kaiserslautern, Germany, Oct., 2008. New York: ACM Press.

Wieringa,R.; Maiden, N.; Mead, N.; Rolland, C. 2005. Requirements engineering paper classification and evaluation criteria: a proposal and a discussion. *Requirements Engineering*. 11(1), pp.102-107.

Winkler, D.; Biffl, S.; Thurnher, B. 2005. Investigating the impact of active guidance on design inspection. In; International Conference on Product Focused Software Process Improvement, Oulu, Finland, Jun., 2005 Berlin: Springer-Verlag.

Wojcicki, M.; Strooper, P. 2006. Maximising the information gained from an experimental analysis of code inspection and static analysis for concurrent java components. In: *In: ACM/IEEE International Symposium on Empirical Software Engineering,* Redondo Beach, USA, Aug., 2006. New York: ACM Press.

Zamboni, A. B.; Thommazo, A. D.; Hernandes, E. C. M.; Fabbri, S. C. P. F. StArt Uma Ferramenta Computacional de Apoio à Revisão Sistemática. In: *Brazilian Conference on Software: Theory and Practice - Tools session*. UFBA.