Static Analysis Techniques and Tools: A Systematic Mapping Study

Vinícius Rafael Lobo de Mendonça
Instituto de Informática
Universidade Federal de Goiás, UFG
Goiânia-GO, Brazil
e-mail: viniciusmendonca@inf.ufg.br

Cássio Leonardo Rodrigues
Instituto de Informática
Universidade Federal de Goiás, UFG
Goiânia-GO, Brazil
e-mail: cassio@inf.ufg.br

Fabrízzio Alphonsus A. de M. N. Soares
Instituto de Informática
Universidade Federal de Goiás, UFG
Goiânia-GO, Brazil
e-mail: fabrizzio@inf.ufg.br

Auri Marcelo Rizzo Vincenzi
Instituto de Informática
Universidade Federal de Goiás, UFG
Goiânia-GO, Brazil
e-mail: auri@inf.ufg.br

***Abstract*—The main disadvantage of static analysis tools is their high false positive rates. False positives are errors that either do not exist or do not lead to serious software failures. Thus, the benefits of automated static analysis tools are reduced due to the need for manual interventions to assess true and false positive warnings. This paper presents a systematic mapping study to identify current state-of-the-art static analysis techniques and tools as well as the main approaches that have been developed to mitigate false positives.**

Keywords-automatic static analysis; false positive; systematic mapping study.

I. Introduction

There are two Verification and Validation (V&V) approaches: dynamic and static [1]. The first approach is characterized by software implementation and defect detection through assessment of program outputs, which is similar to software testing. In the second approach, program execution is not required, and identification of potential faults is carried out through evaluation of software source codes, design diagrams, requirements, etc. Inspections and code reviews are types of techniques used in static analysis but, in general, they are performed by human. The focus of this work is the evaluation of techniques and tools used to perform automated static analysis.

Automated static analysis vocabulary includes the following terms: false positives, true positives and false negatives. A false positive occurs when a tool alerts to the presence of a non-existent fault. A false negative occurs when a fault exists, but it is not detected due to the fact that static analysis tools are not perfectly accurate and may not detect all errors. Finally, a true positive occurs when a tool produces a warning to indicate the presence of a real defect in the product under analysis.

Examples of automated static analysis tools are FindBugs [2], PMD [3] and CheckStyle [4]. The disadvantage of these tools is that they produce a high rate of false positives, whereas developers are only interested in true errors, which are the ones that require correction. False positive alerts lead to an increase in process costs, because their detection is usually done by human intervention. This consumes precious time that could be used for the correction of real faults [5]. Regardless of such disadvantage, static analysis tools are very useful for carrying out initial verification and validation activities compared to other quality assurance procedures, especially due to their low implementation cost.

We conducted a Systematic Mapping Study (SMS) of static analysis techniques and tools to investigate how they avoid false positives. A comprehensive data extraction process and classification of the primary studies provided answers to our research questions. The remaining of this paper is organized as follows: Section II describes the methodology used to conduct the systematic mapping and shows the results obtained in each phase. Section III reveals main results and answers to the research questions defined in Section II. Finally, Section IV shows our conclusions and implications for future studies.

II. Background

This paper shows the development of a systematic map based on the process presented by Petersen [6]. It is composed of the following steps: i) definition of research questions, ii) analysis of relevant studies, iii) study selection, iv) keywording of abstracts, v) data extraction, and vi) mapping process (Figure 1). To increase the reliability of our proposed SMS, some of the guidelines provided by Kitchenham et al. [7] were followed, such as the use of control studies to assess search string quality and Quality Assessment Strategy [8].

A. Research Questions

Our systematic mapping identifies relevant papers on static analysis. We aim to understand the behaviour of automated static analysis tools as well as find out which of the proposed methods mitigate false positives, if any.

 ISBN: 978-1-61208-304-9

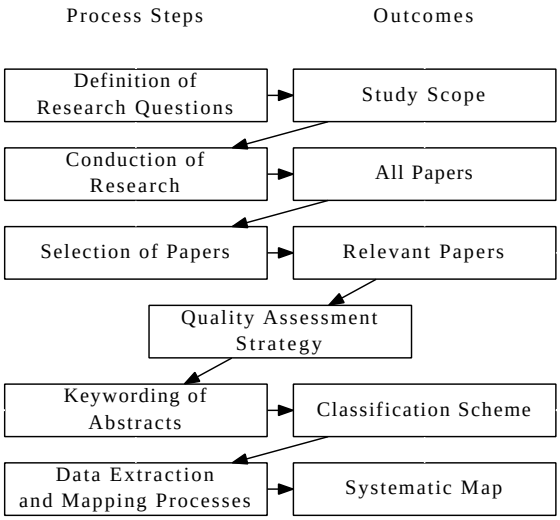

Figure 1. Steps of a Systematic Mapping Study - Adapted from [6].

Each question is answered according to the following criteria: Population, Intervention, Control and Results. The criteria comparison is not applicable to this research, so it was not used. Information on each of them is found in [7]. Our research questions are:

- RQ_1: Which static analysis tools and approaches are used to reduce false positives?
- RQ_2: Which types of warnings are emitted by the tools and which static analysis methods are employed?
- RQ_3: Should various static analysis techniques be used in combination to reduce false positives?

B. Search Strategy, Data Sources and Study Selection

The search strategy involved the creation of a string to operate upon scientific digital libraries for the selection of primary studies. The digital libraries selected were: ACM, IEEE, Engineering Village (Compendex) and SpringerLink. The search string presented in Figure 2 was applied to search engines by using the advanced search mechanism and, in some cases, it was tailored for a specific search engine. The papers used as controls are listed in Table I, it was developed from generic terms found in the control articles used. Static analysis does not have a broad vocabulary, similar terms are usually found in papers, resulting in small search string. The general search string is as follows:

```
((static analysis OR bug finding OR static code analysis OR find bug
OR analysis static) AND (false positive OR warnings))
```

Figure 2. The general search string.

After excluding control articles and their repeated retrievals, the string used in databases returned 615 primary studies, all of them catalogued in a specialized tool, named State of the Art through Systematic Review (StArt) [9]. The results extracted from StArt revealed that ACM retrieved the highest rate of primary studies (52%), followed by Engineering Village (23%), IEEE (17%) and SpringerLink (8%). The main data retrieved from each primary study was stored using the JabRef tool [10] for further classification.

TABLE I. CONTROL PAPERS

#	Title	Citation	Consultation Database
CA_1	*Which Warnings Should I Fix First?*	[11]	ACM
CA_2	*Finding Bugs is Easy*	[5]	ACM
CA_3	*Comparing Bug Finding Tools with Reviews and Tests*	[12]	SpringerLink
CA_4	*A Comparison of Bug Finding Tools for Java*	[13]	IEEE
CA_5	*Static Code Analysis*	[14]	IEEE

The inclusion criteria for this study are:

- IC_1: Primary studies analysing the warnings emitted by static analysis tools;
- IC_2: Primary studies proposing methods/tools to reduce false positives;
- IC_3: Primary studies comparing static analysis tools/methods;

The exclusion criteria are:

- EC_1: Primary studies on static analysis that do not assess warnings or reduce false positives;
- EC_2: Primary studies that are repeated retrievals or contain a maximum of two pages;
- EC_3: Primary studies that cannot be accessed;
- EC_4: Primary studies that do not use static analysis;
- EC_5: Primary studies that are not written in English or Portuguese.

The primary studies underwent two other stages of selection. In the second stage, EC_2, EC_3 and EC_5 were applied. This action ensured a significant reduction in the number of studies, after which 270 studies remained. In the third and last stage, each study's title and abstract were assessed by applying EC_1 and EC_4, eliminated respectively 40% and 33% of the primary studies, remaining 64 papers for quality assessment strategy (Figure 3).

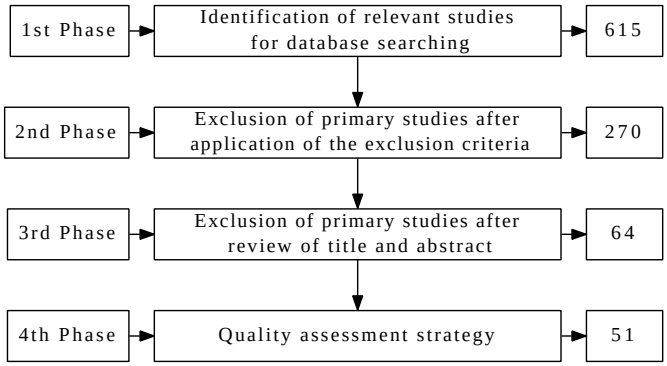

Figure 3. Four stages for selection of primary studies

 ISBN: 978-1-61208-304-9

C. Quality Assessment Strategy and Classification of Selected Studies

After application of the previous steps, 64 papers were selected. Aiming at increasing and ensuring the quality of the developed work, we decided to assess the quality of the selected primary studies based on the criteria created by Dybå and Dingsøyr [8]. These criteria are composed of eleven question but we use only eight since three of them are out of the scope of our work (questions 5, 6 and 9). Information on each of them is found in [8]. It was assigned 1 when the primary study satisfies the criterion and 0 otherwise. Observe that criteria 1 and 2 are considered exclusion criteria since a primary study not satisfying them implies it should be dropped of. Table II presents the final results of the quality assessment. Among 51 articles, 9 (17%) received the maximum score (8) indicating that only a few number of primary studies are concerned to document the stages of the development of their research. 90% of the primary studies collected data to answer the research question, but only 40% of the work are worried to validate the collected data, and 41% are worried to show the results clearly. Observe that the lack of such information makes difficult to draw conclusions about these research areas. Primary studies were classified into three classes, identified after reading the 51 remaining papers: research type, approach type, and types of false positive errors.

In the first class, works were classified based on the types of research: validation research, evaluation research, solution proposal, philosophical papers, opinion papers and experience papers. Information on each of them is found in [6]. The second class was defined to answer RQ_1. Therefore, a classification scheme was developed based on the terminology used in the primary studies. The types of approaches identified were: comparative study, algorithm, new tool, improve existing tool, ranking error, hybrid approach, technique and method. A main approach type was defined for each primary study, which means that it may or may not contain features of other less relevant types of approaches. The characteristics of each approach type are:

- Comparative Study (CS): A primary study that compares static analysis approaches to identify cases in which application of an approach is better than another;
- Algorithm (A): A primary study that proposes a new algorithm that may or may not be used in combination with a static analysis tool to reduce false positives;
- New Tool (TL): A primary study that proposes a new tool which may be more effective than existing tools or used in combination with other tools to identify false positives not yet detected;
- Improve Existing Tool (IT): A primary study that proposes an improvement of an existing tool. For instance, a new bug pattern;
- Ranking Error (RE): A primary study that proposes a new ranking or an improved technique to rank error reports;
- Hybrid Approach (HA): A primary study that combines static and dynamic approaches to reduce the disadvantages of each technique aiming at false positive mitigation;
- Technique (T): A primary study that aims to find a solution to a specific problem [15];
- Method (M): A primary study that searches for a general solution to a problem [15].

The third class is related to false positive errors. It was developed to identify the primary focus of our study. Errors were classified as interface fault, data fault, cosmetic fault, initialization fault, control fault, and computation fault. This classification was created by Basili and Selby [16], where one can find more information on each class of fault. However, most of the 64 selected studies mention the false positives generated by static analysis tools in a general way. Thus, a new error category was created: extensive study. The primary studies that identify more than 10 error types were also included in this category. Studies containing 2-9 defects were classified into several categories. Also, when more than one defect of the same type was detected, only one would be considered. Most primary studies aimed at reducing false positives of various types. This is a relevant result, because the purpose of our systematic mapping is to provide an overview of the research field.

D. Data Extraction and Mapping Processes

The JabRef and spreadsheets were used in the remaining 51 primary studies to create a three-class classification: research type, error type, and approach type. Figure 4 shows that 25 out of 51 (49%) primary studies were classified as solution proposals. This indicates that many proposed new approaches or improvements for existing ones aimed at reducing false positive alerts emitted by static analysis tools. However, none of them was classified as validation research, indicating that there is no experimental validation of the proposed approaches.

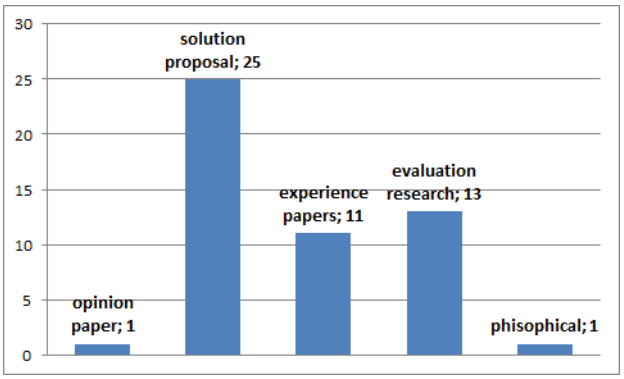

Figure 4. The classification of selected papers in relation research classes

Regarding the types of approaches identified, there were a variety of proposed solutions to problems, mainly methods (18.91%) and new tools (16.21%), which are shown in Figure 5). This indicates that many researchers, who were not satisfied with existing tools, proposed new ones as well as prototypes to assist developers. Some works also presented

 ISBN: 978-1-61208-304-9

TABLE II. RESULTS OF SYSTEMATIC MAPPING STUDY QUALITY EVALUATION AND THE AMOUNT OF THE FALSE POSITIVES ACHIEVED IN EACH PRIMARY STUDY

Paper	RE	AI	CO	RD	DC	DA	FI	VA	RS	TFP	A	FPR(%)
[17]	1	1	1	1	1	0	1	0	6	S	T	63%
[18]	1	1	1	1	1	0	0	1	6	ND	T	0%
[19]	1	1	1	1	1	1	1	1	8	S	T	20%
[20]	1	1	1	1	1	0	0	1	6	S	T	BCI 13%
[21]	1	1	1	1	1	0	0	1	6	S	T	IINC
[22]	1	1	1	1	1	1	1	1	8	S	RE	19%
[23]	1	1	1	1	1	1	1	1	8	S	RE	BCI 0%
[24]	1	1	0	1	1	0	0	1	5	S	RE	IINC
[25]	1	1	1	1	1	0	1	1	7	S	M	IINC
[26]	1	1	0	1	1	0	1	1	6	SB	M	0%
[27]	1	1	0	1	1	1	1	1	7	S	M	IINC
[28]	1	1	1	1	1	1	0	1	7	S	M	IINC
[29]	1	1	1	1	1	1	1	0	7	S	M	32%
[30]	1	1	1	1	1	1	1	1	8	S	M	BCI 74%
[31]	1	1	0	1	1	0	1	1	6	S	M	IINC
[32]	1	1	0	1	1	1	1	0	6	S	M	BCI 0%
[33]	1	1	1	1	1	1	0	1	7	S	M	IINC
[34]	1	1	0	1	0	0	0	0	3	S	M	34%
[35]	1	1	0	1	1	1	1	1	7	C	M	IINC
[36]	1	1	1	1	0	0	1	0	5	S	IT	15%
[37]	1	1	1	1	1	1	1	1	8	S	IT	IINC
[38]	1	1	1	1	0	0	0	1	5	S	IT	BCI 0%
[39]	1	1	0	1	1	0	0	1	5	S	IT	IINC
[40]	1	1	1	1	1	1	1	1	8	S	IT	IINC
[41]	1	1	0	1	1	1	1	1	7	S	TL	BCI 0%
[42]	1	1	0	1	1	0	0	0	4	S	TL	IINC
[43]	1	1	1	1	1	0	0	0	5	IB	TL	IINC
[44]	1	1	1	1	1	1	1	1	8	BO	TL	IINC
[45]	1	1	1	0	1	0	0	1	5	S	TL	IINC
[46]	1	1	0	1	1	0	0	0	4	S	TL	IINC
[47]	1	1	1	1	1	1	1	1	8	C	TL	IINC
[48]	1	1	1	1	1	0	0	0	5	D	TL	IINC
[49]	1	1	0	1	1	0	0	0	4	S	TL	IINC
[50]	1	1	0	1	1	1	0	0	5	S	TL	IINC
[51]	1	1	0	1	1	0	0	0	4	CS		
[52]	1	1	0	1	1	0	0	0	4	CS		
[53]	1	1	0	1	1	0	0	1	5	CS		
[54]	1	1	0	1	0	0	0	0	3	CS		
[55]	1	1	1	1	1	0	0	0	5	CS		
[56]	1	1	0	1	1	0	1	0	5	CS		
[57]	1	1	1	0	1	0	0	1	5	DR	A	BCI 0 %
[58]	1	1	0	1	1	0	0	1	5	DR	A	IINC
[59]	1	1	1	1	1	0	0	1	6	S	A	IINC
[60]	1	1	0	1	1	0	0	0	4	S	A	BCI 20%
[61]	1	1	0	1	1	1	0	1	6	S	A	IINC
[62]	1	1	1	1	1	0	0	0	5	S	A	BCI 38%
[63]	1	1	0	0	1	0	0	1	4	S	HA	IINC
[64]	1	1	0	1	1	1	0	0	6	S	HA	IINC
[65]	1	1	1	1	1	1	0	0	6	S	HA	IINC
[66]	1	1	1	1	1	1	0	1	7	S	HA	IINC
[67]	1	1	0	1	1	0	0	0	4	S	HA	IINC
RS	51	51	28	49	46	20	21	32	*	*	*	*

Acronyms

RE: RESEARCH	VA: VALUE	SB: STRING BUG
AI: AIM	RS: RESULT	C: CONCURRENCE
CO: CONTEXT	TFP: TYPE OF FALSE POSITIVE	IB: INTEGER BUG
RD: RESEARCH DESIGN	A: APPROACH	BO: BUFFER OVERFLOW
DC: DATA COLLECTION	FPR: FALSE POSITIVE RATE	RC: DATA RACE
DA: DATA ANALYSIS	S: SEVERAL	BCI: BEST CASE IS
FI: FINDINGS	ND: NULL DEFERENCE	IINC: IT IS NOT CLEAR

improvements for the FindBugs tool [2], which is used for static analysis of Java programs. Most primary studies suggest ways to mitigate false positives of various types (Figure 5). The fact that a large number of research proposals focus on the many kinds of errors exposed by static analysis contributes to the provision of a variety of mitigation techniques.

A bubble chart was designed to display the intersection of two classes: approach type and types of false positives (Figure 5). According to Petersen et al. [6], the bubble chart was effective in the sense that it gave an overview of the research field and produced a map of results.

III. MAIN FINDINGS

In this section, we present the answers to the three proposed research questions based on the primary studies found.

A. Answer to RQ_1 – Tools and Approaches

With respect the tools, it can be observed that a significant number of static analysis tools is used by researchers trying to reduce false positives, i.e., there is not a consensus that the tool A is better or worse than the tool B, or whether it is better to use an open source tool or a commercial tool. Some authors believe that the use of tools is not mutually exclusive because if tool A finds more real faults of "null deference" than a tool B, but B is better than A on detecting real faults of "buffer overflow", then the correct choice should be to combine both tools taking the advantage of both. This strategy aims to potentiating the strengths of each tool, increasing the amount of real defects found, and also the precision, because if more than one tool reports a same warning on the same line, this may indicates that the probability of the warning corresponds to a true positive is greater than the one reported by a single tool.

The collected data indicate that 16.21% of the papers propose a new static analysis tool, the majority without any comparison with other existing tool to demonstrate the effectiveness of the proposed approach. From this percentage, just 25% seek to mitigate defects in a comprehensive way, the other 75% remaining seek to reduce false positive of a specific class. This large concentration of new tools with a focus on some specific defect demonstrates a possible deficiency of existing tools for detecting such defects. The main defects found are "data fault" and "initialization fault" representing together 50% of the works proposed of improved tool.

There are also 12.16% of the studies which present hybrid techniques by combining dynamic and static analysis. According to Aggarwal and Jalote [45], IT community believes that the static and dynamic approaches are complementary. Static analyzers are faster and simpler to use than dynamic ones. Moreover, they also help to identify problems earlier in the development process, when the cost to correct them is lower but, in general, static analyzers generate large amounts of false positives and false negatives. On the other hand, dynamic analyzers are accurate and generate few false positives, but to test all possible conditions in a program with thousands of line of code is practically impossible. During the systematic mapping were found works which use both approaches in a complementary way to reduce the drawbacks of each other.

Among the tools used or cited in the primary studies, those that stand out are: FindBugs and Checkstyle. Both are used together with PMD by the quality platform SonarQube for the calculation of part of its static metrics, but SonarQube does not seek to reduce false positives.

B. Answer to RQ_2 – The types of false positive errors

We also tried to identify specific classes of false positives warnings but the majority of the works (33.78%) fell down in the generic classification of false positives, i.e., the information is not available. The fact that significant amount of work develops techniques or tools to mitigate various false positives is something negative demonstrating the existence of a large gap in this research area to be filled. 20% of the proposed methods provided generic methods which are palliative solutions, not solving the problem of the high rate of false positive efficiently.

Table II shows a summary of the type of false positive, the technique and the false positive rate of the 51 primary studies investigated. Excluding the comparative studies that do

 ISBN: 978-1-61208-304-9

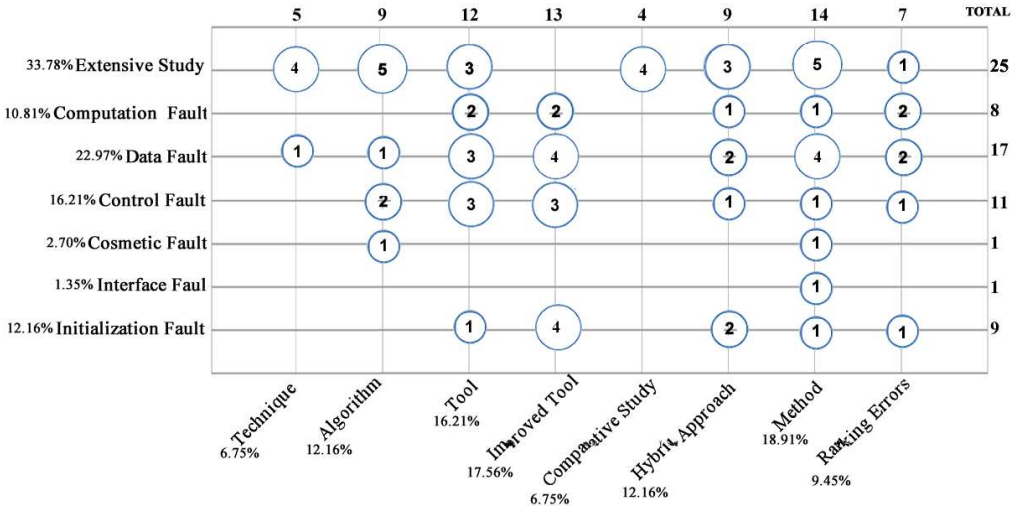

Figure 5. Bubble chart – Types of Approaches × Types of False Positives

not investigate the effectiveness of a specific approach, 63% of the remaining works do not make clear how the proposed approach was efficient, they just mention that the false positive rate was reduced without any evidences. But, how much is the reduction? Without this information, the proposed approach has no change to be addopted on real software development environments. Therefore, improve the way the experiments are conducted on these research area is of fundamental importance to provide such an evidences.

Maybe the real solution does not treats in reduce the rate of false positives, but reducing the amount of certain types of false positives. Treating this problem broadly, your solution may be far from being achieved. One approach might be to assign a weight or priority for each type of false positive, characteristic established in some bug-finding tools (FindBugs, PMD, and Checkstyle are some of them).

In this manner, bugs with greater weight, hence higher priority should be given greater attention and other bugs with lower priority can be observed or subsequently depending on how much low is your priority should be ignored. Currently, we live in a scenario that programs are getting ever larger, exceeding millions of line of code, so the task of analyzing the warnings emitted by bug finding tools, should be performed in a smarter way, reducing the human effort and time on this activity. To do this, we think an important step is to valorize the types of false positives but, among existing tools, there is no consensus between the weight or priority, and similar or identical bugs may have different values in different tools.

C. Answer to RQ_3 – Application of Static Analysis Tools

The answer to reduce false positives rate might be resumed in one word: combine. It is not necessary to create something new or improve something that already exists, but the strategy should be to combine static analysis tools and/or dynamic and static tools and to conduct significant empirical studies to identify the best combination of tools to reduce drawbacks and to increase benefits. Unfortunately, the task is not so easy since there are several technical details to be overcome to create a meta static analysis tool, such as, how to combine different warning's prioritization classification in a single meta tool; how to manage different output formats; and so on.

Rutar et al. [13] is an example of primary study which uses this approach. They utilizes tools in conjunction with applications of different sizes. Each tool performs a different balancing to equilibrate the real location of errors, the generation of false positives and true positives and, consequently, there is little overlap among the generated warnings. These different approaches of balancing can involve the necessity of using multiple tools in the verification of an application. Then, Rutar et al. [13] suggested the development of a meta-tool, which combines the results of different tools for searching errors.

IV. Final Considerations

This paper showed the development of a systematic mapping study on static analysis approaches and tools aiming at reducing the number of false positives generated. After the selection of 51 studies, the mapping combined protocol processes developed by Petersen et al. [6] and Kitchenham et al. [7]. The selected works provided a variety of static analysis approaches, including proposals for improving existing tools. FindBugs stands out in the sense that many primary studies not only use it, but also discuss how it may be improved.

Among the retrieved studies, there was a lack of works on the types of false positive errors and the tools that generate them. This kind of research would help developers identify the tools that best serve their needs. The mapping also revealed studies that use hybrid approaches, which combine static and dynamic analyses techniques. Furthermore, a combination of different static analysis approaches proved more efficient than their isolated use.

Based on the mapping results and due to a lack of validation research on the subject, we propose a large-scale experimental study aiming at finding answers to open questions. This would

 ISBN: 978-1-61208-304-9

contribute to advancements in the use of static analysis tools in the early stages of the development cycle, as well as identification of the types of defects that should be treated by other verification and validation techniques. Moreover, the development of a methodology for combining static analysis approaches and tools is also recommended for future research.

ACKNOWLEDGMENT

The authors would like to thank the Instituto de Informática – INF/UFG, Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – CAPES – Brasil, and Fundação de Amparo à Pesquisa do Estado de Goiás – FAPEG – Brasil, which support this work.

REFERENCES

[1] I. Sommerville, S. Melnikoff, R. Arakaki, and E. de Andrade Barbosa, *Software Engineering*. ADDISON WESLEY BRA, 2008, [retrieved: Oct., 2013]. [Online]. Available: http://books.google.com.br/books?id=ifIYOgAACAAJ

[2] FindBugs, [retrieved: Oct., 2013]. [Online]. Available: http://findbugs.sourceforge.net/

[3] PMD, [retrieved: Oct., 2013]. [Online]. Available: pmd.sourceforge.net/

[4] CheckStyle, [retrieved: Oct., 2013]. [Online]. Available: http://checkstyle.sourceforge.net

[5] D. Hovemeyer and W. Pugh, "Finding bugs is easy," *SIGPLAN Not.*, vol. 39, no. 12, pp. 92–106, 2004.

[6] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson, "Systematic mapping studies in software engineering," in *12th International Conference on Evaluation and Assessment in Software Engineering*, vol. 17, 2008, p. 1.

[7] B. Kitchenham and S. Charters, "Guidelines for performing Systematic Literature Reviews in Software Engineering," Keele University and Durham University Joint Report, Tech. Rep. EBSE 2007-001, 2007.

[8] T. D. Aě and T. D. Aÿyr, "Empirical studies of agile software development: A systematic review," *Information and Software Technology*, vol. 50, pp. 833–859, 2008.

[9] StArt, [retrieved: Oct., 2013]. [Online]. Available: http://lapes.dc.ufscar.br/tools/start-tool

[10] JabRef, [retrieved: Oct., 2013]. [Online]. Available: http://jabref.sourceforge.net/

[11] S. Kim and M. D. Ernst, "Which warnings should i fix first?" in *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, ser. ESEC-FSE'07. New York, NY, USA: ACM, 2007, pp. 45–54.

[12] S. Wagner, J. Jürjens, C. Koller, and P. Trischberger, "Comparing bug finding tools with reviews and tests," in *Proceedings of the 17th IFIP TC6/WG 6.1 international conference on Testing of Communicating Systems*, ser. TestCom'05. Berlin, Heidelberg: Springer-Verlag, 2005, pp. 40–55.

[13] N. Rutar, C. B. Almazan, and J. S. Foster, "A comparison of bug finding tools for java," in *Proceedings of the 15th International Symposium on Software Reliability Engineering*, ser. ISSRE'04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 245–256.

[14] P. Louridas, "Static code analysis," *Software, IEEE*, vol. 23, no. 4, pp. 58–61, 2006.

[15] L. Nascimento, D. Viana, P. Silveira Neto, D. Martins, V. Garcia, and S. Meira, "A systematic mapping study on domain-specific languages," in *ICSEA'12, The Seventh International Conference on Software Engineering Advances*, 2012, pp. 179–187.

[16] V. Basili and R. Selby, "Comparing the effectiveness of software testing strategies," *Software Engineering, IEEE Transactions on*, vol. SE-13, no. 12, pp. 1278–1296, 1987.

[17] M. G. Nanda and S. Sinha, "Accurate interprocedural null-dereference analysis for java," in *Proceedings of the 31st International Conference on Software Engineering*, ser. ICSE'09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 133–143.

[18] P. Emanuelsson and U. Nilsson, "A comparative study of industrial static analysis tools," *Electron. Notes Theor. Comput. Sci.*, vol. 217, pp. 5–21, Jul. 2008.

[19] Lucia, D. Lo, L. Jiang, and A. Budi, "Active refinement of clone anomaly reports," in *Proceedings of the 2012 International Conference on Software Engineering*, ser. ICSE'12. Piscataway, NJ, USA: IEEE Press, 2012, pp. 397–407.

[20] S. S. Heckman, "Adaptively ranking alerts generated from automated static analysis," *Crossroads*, vol. 14, no. 1, pp. 7:1–7:11, Dec. 2007.

[21] S. Heckman and L. Williams, "A model building process for identifying actionable static analysis alerts," in *Proceedings of the 2009 International Conference on Software Testing Verification and Validation*, ser. ICST'09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 161–170.

[22] A. Vetro, M. Morisio, and M. Torchiano, "An empirical validation of findbugs issues related to defects," Durham University – Grey College, Durham, Apr., pp. 144–153.

[23] E. Bodden and K. Havelund, "Aspect-oriented race detection in java," *IEEE Trans. Softw. Eng.*, vol. 36, no. 4, pp. 509–527, Jul. 2010.

[24] G. Liang, L. Wu, Q. Wu, Q. Wang, T. Xie, and H. Mei, "Automatic construction of an effective training set for prioritizing static analysis warnings," in *Proceedings of the IEEE/ACM international conference on Automated software engineering*, ser. ASE'10. New York, NY, USA: ACM, 2010, pp. 93–102.

[25] S. Kim, T. Zimmermann, K. Pan, and E. J. J. Whitehead, "Automatic identification of bug-introducing changes," in *Proceedings of the 21st IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE'06. Washington, DC, USA: IEEE Computer Society, 2006, pp. 81–90.

[26] P. Chen, Y. Wang, Z. Xin, B. Mao, and L. Xie, "Brick: A binary tool for run-time detecting and locating integer-based vulnerability," in *Availability, Reliability and Security, 2009. ARES'09. International Conference on*, 2009, pp. 208–215.

[27] D. Babi? and A. J. Hu, "Calysto: Scalable and precise extended static checking," 2008.

[28] C. Csallner and Y. Smaragdakis, "Check 'n' crash: combining static checking and testing," in *Proceedings of the 27th international conference on Software engineering*, ser. ICSE'05. New York, NY, USA: ACM, 2005, pp. 422–431.

[29] A. Fehnker, R. Huuck, and S. Seefried, "Concurrency, compositionality, and correctness," D. Dams, U. Hannemann, and M. Steffen, Eds. Springer-Verlag, 2010, ch. Counterexample guided path reduction for static program analysis, pp. 322–341.

[30] F. Ivancic, G. Balakrishnan, A. Gupta, S. Sankaranarayanan, N. Maeda, H. Tokuoka, T. Imoto, and Y. Miyazaki, "Dc2: A framework for scalable, scope-bounded software verification," in *Automated Software Engineering (ASE), 2011 26th IEEE/ACM International Conference on*, 2011, pp. 133–142.

[31] S. Kim, H. Zhang, R. Wu, and L. Gong, "Dealing with noise in defect prediction," in *Proceedings of the 33rd International Conference on Software Engineering*, ser. ICSE '11. New York, NY, USA: ACM, 2011, pp. 481–490.

[32] S. Lu, S. Park, and Y. Zhou, "Detecting concurrency bugs from the perspectives of synchronization intentions," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 23, no. 6, pp. 1060–1072, 2012.

[33] A. Tomb and C. Flanagan, "Detecting inconsistencies via universal reachability analysis," in *Proceedings of the 2012 International Symposium on Software Testing and Analysis*, ser. ISSTA'12. New York, NY, USA: ACM, 2012, pp. 287–297.

[34] A. C. Nguyen and S.-C. Khoo, "Discovering complete api rules with mutation testing," in *Mining Software Repositories (MSR), 2012 9th IEEE Working Conference on*, 2012, pp. 151–160.

[35] J. Hoenicke, K. R. Leino, A. Podelski, M. Schäf, and T. Wies, "Doomed program points," *Form. Methods Syst. Des.*, vol. 37, no. 2-3, pp. 171–199, Dec. 2010.

[36] C. Csallner, Y. Smaragdakis, and T. Xie, "Dsd-crasher: A hybrid analysis tool for bug finding," *ACM Trans. Softw. Eng. Methodol.*, vol. 17, no. 2, pp. 8:1–8:37, May 2008.

[37] B. Chimdyalwar and S. Kumar, "Effective false positive filtering for evolving software," in *Proceedings of the 4th India Software Engineering Conference*, ser. ISEC'11. New York, NY, USA: ACM, 2011, pp. 103–106.

[38] H. Shen, J. Fang, and J. Zhao, "Efindbugs: Effective error ranking for findbugs," in *Software Testing, Verification and Validation (ICST), 2011 IEEE Fourth International Conference on*, 2011, pp. 299–308.

 ISBN: 978-1-61208-304-9

[39] A. Shi and G. Naumovich, "Field escape analysis for data confidentiality in java components," in *Software Engineering Conference, 2007. APSEC 2007. 14th Asia-Pacific*, 2007, pp. 143–150.

[40] Y. Kim, J. Lee, H. Han, and K.-M. Choe, "Filtering false alarms of buffer overflow analysis using smt solvers," *Inf. Softw. Technol.*, vol. 52, no. 2, pp. 210–219, Feb. 2010.

[41] E. Bodden, P. Lam, and L. Hendren, "Finding programming errors earlier by evaluating runtime monitors ahead-of-time," in *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, ser. SIGSOFT'08/FSE-16. New York, NY, USA: ACM, 2008, pp. 36–47.

[42] F. Otto and T. Moschny, "Finding synchronization defects in java programs: extended static analyses and code patterns," in *Proceedings of the 1st international workshop on Multicore software engineering*, ser. IWMSE'08. New York, NY, USA: ACM, 2008, pp. 41–46.

[43] Q. Chen, L. Wang, and Z. Yang, "Heat: An integrated static and dynamic approach for thread escape analysis," in *Computer Software and Applications Conference, 2009. COMPSAC'09. 33rd Annual IEEE International*, vol. 1, 2009, pp. 142–147.

[44] A. Shi and G. Naumovich, "Improving data integrity with a java mutability analysis." in *APSEC*. IEEE Computer Society, 2007, pp. 135–142.

[45] A. Aggarwal and P. Jalote, "Integrating static and dynamic analysis for detecting vulnerabilities," in *Computer Software and Applications Conference, 2006. COMPSAC'06. 30th Annual International*, vol. 1, 2006, pp. 343–350.

[46] D. Kong, Q. Zheng, C. Chen, J. Shuai, and M. Zhu, "Isa: a source code static vulnerability detection system based on data fusion," in *Proceedings of the 2nd international conference on Scalable information systems*, ser. InfoScale '07, 2007, pp. 55:1–55:7.

[47] M. G. Nanda, M. Gupta, S. Sinha, S. Chandra, D. Schmidt, and P. Balachandran, "Making defect-finding tools work for you," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 2*, ser. ICSE'10. New York, NY, USA: ACM, 2010, pp. 99–108.

[48] M. Al-Ameen, M. Hasan, and A. Hamid, "Making findbugs more powerful," in *Software Engineering and Service Science (ICSESS), 2011 IEEE 2nd International Conference on*, 2011, pp. 705–708.

[49] C. Le Goues and W. Weimer, "Measuring code quality to improve specification mining," *Software Engineering, IEEE Transactions on*, vol. 38, pp. 175–190, 2012.

[50] P. Anderson, "Measuring the value of static-analysis tool deployments," *Security Privacy, IEEE*, vol. 10, pp. 40–47, 2012.

[51] S. Kim, K. Pan, and E. E. J. Whitehead, Jr., "Memories of bug fixes," in *Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering*, ser. SIGSOFT'06/FSE-14, 2006, pp. 35–45.

[52] C. Cifuentes and B. Scholz, "Parfait: designing a scalable bug checker," in *Proceedings of the 2008 workshop on Static analysis*, ser. SAW'08, 2008, pp. 4–11.

[53] Z. Ding, H. Wang, and L. Ling, "Practical strategies to improve test efficiency," *Tsinghua Science and Technology*, vol. 12, pp. 250–254, 2007.

[54] V. Pessanha, R. J. Dias, J. a. M. Lourenço, E. Farchi, and D. Sousa, "Practical verification of high-level dataraces in transactional memory programs," in *Proceedings of the Workshop on Parallel and Distributed Systems: Testing, Analysis, and Debugging*, ser. PADTAD'11. New York, NY, USA: ACM, 2011, pp. 26–34.

[55] S. Heckman and L. Williams, "On establishing a benchmark for evaluating static analysis alert prioritization and classification techniques," in *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*, ser. ESEM'08, 2008, pp. 41–50.

[56] S. Kim and M. D. Ernst, "Prioritizing warning categories by analyzing software history," in *Proc. of Int'l Workshop on Mining Software Repositories (MSR'2007*, 2007, p. 27.

[57] K. Li, C. Reichenbach, C. Csallner, and Y. Smaragdakis, "Residual investigation: predictive and precise bug detection," in *Proceedings of the 2012 International Symposium on Software Testing and Analysis*, ser. ISSTA'12. New York, NY, USA: ACM, 2012, pp. 298–308.

[58] D. Reimer, E. Schonberg, K. Srinivas, H. Srinivasan, B. Alpern, R. D. Johnson, A. Kershenbaum, and L. Koved, "Saber: smart analysis based error reduction," *SIGSOFT Softw. Eng. Notes*, vol. 29, pp. 243–251, 2004.

[59] J. Wu, Y. Tang, G. Hu, H. Cui, and J. Yang, "Sound and precise analysis of parallel programs through schedule specialization," in *Proceedings of the 33rd ACM SIGPLAN conference on Programming Language Design and Implementation*, ser. PLDI'12. New York, NY, USA: ACM, 2012, pp. 205–216.

[60] D. Babić, L. Martignoni, S. McCamant, and D. Song, "Statically-directed dynamic automated test generation," in *Proceedings of the 2011 International Symposium on Software Testing and Analysis*, ser. ISSTA'11, New York, NY, USA, 2011, pp. 12–22.

[61] W. Han, M. Ren, S. Tian, L. Ding, and Y. He, "Static analysis of format string vulnerabilities," in *Software and Network Engineering (SSNE), 2011 First ACIS International Symposium on*, 2011, pp. 122–127.

[62] W. H. K. Bester, C. P. Inggs, and W. C. Visser, "Test-case generation and bug-finding through symbolic execution," in *Proceedings of the South African Institute for Computer Scientists and Information Technologists Conference*, ser. SAICSIT'12. New York, NY, USA: ACM, 2012, pp. 1–9.

[63] A. Avancini and M. Ceccato, "Towards security testing with taint analysis and genetic algorithms," in *Proceedings of the 2010 ICSE Workshop on Software Engineering for Secure Systems*, ser. SESS'10. New York, NY, USA: ACM, 2010, pp. 65–71.

[64] S. Keul, "Tuning static data race analysis for automotive control software," in *Source Code Analysis and Manipulation (SCAM), 2011 11th IEEE International Working Conference on*, 2011, pp. 45–54.

[65] N. Ayewah and W. Pugh, "Using checklists to review static analysis warnings," in *Proceedings of the 2nd International Workshop on Defects in Large Software Systems: Held in conjunction with the ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2009)*, ser. DEFECTS'09. New York, NY, USA: ACM, 2009, pp. 11–15.

[66] J. Lawall, J. Brunel, N. Palix, R. Hansen, H. Stuart, and G. Muller, "Wysiwib: A declarative approach to finding api protocols and bugs in linux code," in *Dependable Systems Networks, 2009. DSN '09. IEEE/IFIP International Conference on*, 2009, pp. 43–52.

[67] T. Kremenek and D. Engler, "Z-ranking: Using statistical analysis to counter the impact of static analysis approximations," in *Static Analysis*, ser. Lecture Notes in Computer Science, R. Cousot, Ed. Springer Berlin Heidelberg, 2003, vol. 2694.

 ISBN: 978-1-61208-304-9