Overview of the advantages and disadvantages of static code analysis tools

Liisa Sakerman
Institute of Computer Science
University of Tartu
Tartu, Estonia
liisa.sakerman@ut.ee

Rain Hallikas
Institute of Computer Science
University of Tartu
Tartu, Estonia
rain.hallikas@ut.ee

***Abstract*—There are numerous tools available for developers that analyse the source code to find vulnerabilities and help improve the quality as well as maintainability of the code. Oftentimes the usage of these static analysis tools can also bring upon different problems. The aim of this systematic mapping study is to give an overview of the advantages and limitations of static analysis tools, introduce how these limitations could be overcome and gather info on more popular tools.**

Keywords—static code analysis tools, systematic mapping study

I. INTRODUCTION

Software maintenance is a continuous process present in the pre-delivery as well as in the post-delivery stage of a software lifecycle [1]. As it is one of the more expensive parts of the product's development, it is important to develop well-maintainable code [2]. One way to ensure this is conducting code analysis. Code analysis can be static or dynamic: for static analysis the run-through is performed without the actual execution of the program, as opposed to dynamic. Oftentimes the static approach is better suited for code analysis, as it can be performed without the whole product fully ready or even compliable [3]. Static analysis can be manual or automatic: manual ways include for example program comprehension or code review [3]. These however require the human-factor, which is expensive, error-prone and time-consuming [3]. Therefore, static analysis tools are a popular way of conducting the automated review of the source code. For this reason, it was decided to research the different static analysis tools, their advantages and disadvantages and ways of overcoming these limitations.

II. METHODOLOGY

This paper is a systematic map and is composed of the following steps: i) definition of research questions, ii) data search, iii) study selection and quality assessment, iv) data extraction and v) results.

A. Research Questions

This systematic mapping aims to identify relevant papers on static code analysis tools. The goal is to give an overview of the advantages and limitations of static analysis tools and how these limitations can be overcome. This leads to the following research questions (RQs):

- RQ1: What are the main advantages to the usage of static analysis tools?
- RQ2: What are the main disadvantages to the usage of static analysis tools?
- RQ3: Which static analysis tools are more mentioned in relevant studies in the last 10 years?
- RQ4: How can the problems related to the static analysis tools be minimized?

B. Search Strategy and Data Sources

The search strategy involved the creation of the search string for the database search. The digital library chosen for this study is Ebsco Discovery Service. This database includes results from numerous different digital libraries such as IEEE Xplore and Scopus and was so chosen for the main search pool. The search string was as follows:

> (static tool analysis) OR (static code analysis) OR (static analysis tools) OR (source code analysis) OR (static tool analysers) OR (static code analysers) OR (source code analysers)

Fig. 1. The search string.

As the topic of static analysis does not have a very broad vocabulary, the resulting search query is simplistic for the most accurate results.

This string returned 122 722 results with no additional filtering. To first limit these results, the query search is done on only the publications' titles. Resulting pool is 4041, which will be further narrowed down with exclusion and inclusion criteria.

C. Study Selection and Quality Assessment

Exact duplicates were removed from the search list by Ebsco Discovery Service itself. The following exclusion criteria was set:

- EC1: Papers published later than 2010 (2414 remaining)
- EC2: Papers written in some other language than English (1888 remaining)
- EC3: Papers that are not conference materials or academic journals (1543 remaining)
- EC4: Papers that do not have the full text publicly available (1180)
- EC5: Papers that do not contain the subject „tools" (65 remaining)

The following inclusion criteria was set:

- IC1: Papers mentioning static analysis tools in use.
- IC2: Papers discussing the advantages of static analysis tools.

- IC3: Papers discussing problems related to static analysis tool.
- IC4: Papers suggesting solutions to solving problems related to static analysis tools (19 remaining)

To the 4041 results found by the search string from the database, EC1 was applied to eliminate older studies and 2414 studies remained after that. Then EC2 was applied, and 1888 papers remained. After that, EC3 was applied to exclude all possibly less relevant publications and get higher quality papers – 1543 results remained. EC4 was applied next and with 1180 results remaining after that, EC5 was applied to further ensure that the publications found also include text on static analysis tools and not only just static analysis. 65 papers remained after applying all the exclusion criteria. Those papers' titles and abstracts were read through and analysed according to the set inclusion criteria and from there 19 papers were chosen. Additionally, 4 more publications were added to this list, that were found by manual search in the process of writing the introduction and collecting background info). This means collectively 23 publications were found from the study selection.

Additional quality criteria were used to assess the quality of the studies:

- QC1: Are the aim, methodology and results of the study clearly stated?

No more papers were excluded after the assessment of the quality criteria.

D. Data Extraction

The 23 papers were read through and the info gotten from them was gathered into a data extraction table. Data items, that were looked for in addition to article title and author names, were static analysis tool advantages, static analysis tool disadvantages, static analysis tools and solutions. It was collected, which advantages do these publications mention in regard to static analysis tools, and similarly, which disadvantages the papers bring up. Additionally, it was sought, which tools do the papers mention (either their use or when citing some other study). Finally, the publications' potential recommendations to the tools' limitations were also analysed.

TABLE I. DATA EXTRACTION FORM

Data Item	Value	RQ
Article title	Name of the article	
Author name(s)	Set of names of the authors	
Static analysis tool advantages	Advantages mentioned	RQ1
Static analysis tool disadvantages	Disadvantages mentioned	RQ2
Static analysis tools	Tools mentioned	RQ3
Solutions	Solutions mentioned to the disadvantages	RQ4

III. MAIN FINDINGS

A. RQ1 – Advantages

TABLE II. ADVANTAGES

General advantage	Studies
Tools help to detect defects without execution (early detection).	[4], [6], [8], [10], [12], [14], [15], [16], [20], [22], [23]
Understand the behaviour of a program.	[4], [8], [9], [11], [18], [24], [25]
Using tools is less time-consuming than manual static code analysis.	[4], [5], [11], [12], [24], [26]
Tools can detect more advanced but common problems (i.e., memory leaks, buffer overflow, race condition, security issues).	[10], [13], [19], [22], [23], [24]
Tools help to keep code at high, long-lasting quality and contain the best practice patterns.	[10], [11], [16], [18]
Tools help to reduce code review effort and produce no code review comments in the code.	[11], [12], [24]
Some tools are highly customizable to specific needs.	[5]
Easier to achieve full code coverage.	[23]
Wide range of tools to choose from.	[24]
Tools are becoming increasingly easier to use.	[25]

Table 2 presents the general advantages of static analysis tools found across 23 publications to answer RQ1. The advantages are sorted by occurrence, starting with the most popular advantage. The advantages with different wording but similar ideas have been amassed under a single description.

B. RQ2 – Disadvantages

TABLE III. DISADVANTAGES

General disadvantage	Studies
Tools can have false positive results.	[6], [9], [11], [12], [14], [16], [17], [18], [20], [21], [25], [26]
Tools can have false negative results which usually leads to human involvement.	[5], [7], [9], [10], [12], [14], [25]
A single tool might not be able to detect all defects.	[4], [6], [8], [10], [13], [23]
Unclear and different warning messages among tools.	[5], [16], [17], [20], [22]
Complex defects and defect types lack an efficient tool.	[6], [10], [13], [26]
Difficult to integrate some tools because they rely on older versions of libraries of compilers.	[6], [16], [26]
Defects must be visible in static source code.	[4], [23]

The tools are not that helpful as usually only a small fraction of issues is fixed by developers.	[18], [22]
Not all programming languages are supported for specific tools (and in general).	[4]
Some tools have high memory consumption.	[5]
Complex control flow or data flow reduces tools' success rate.	[14]
Some tools have low customizability.	[20]

Table 3 presents the general disadvantages to answer RQ2, similarly to advantages in Table 2. The disadvantages are sorted by occurrence and similar ideas have been gathered under one detail.

C. RQ3 – Tools

TABLE IV. TOOLS

Tool	Studies
FindBugs	[4], [7], [11], [12], [15], [17], [18], [19], [20], [24], [25], [26]
PMD	[4], [7], [11], [15], [17], [24], [25]
Cppcheck	[4], [6], [10], [13], [17]
SonarQube	[4], [15], [18], [24], [25]
CheckStyle	[7], [24], [25], [26]
Splint	[4], [6], [10]
Flawfinder	[4], [6], [10]
Coverty	[5], [10], [25]
FindSecBugs	[19], [21], [22]
Parasoft	[5], [10]
Sparse	[6], [10]
Clang	[6], [13]

Table 4 presents the tools found from the 23 publications to answer RQ3. It contains only tools that were mentioned at least twice. The tools mentioned once were: NDepend, Nitriq, ReSharper, PVS-studio, Visual Code Grepper, Frama-C, Oclint, Infer, Uno, Flint++, Apache-rat, Clirr, jDepend, VCG, Polyspace, CSTAT, QAC, Astree, Klocwork, YASCA, ITS4, Goanna, RATS, CAT.NET, Sourceinsigthscan, Understand, SpotBugs, Better Code Hub, StyleCop, Gendarme.

D. RQ4 – Solutions

TABLE V. SOLUTIONS

Tool	Studies
Combining tools can help detect more defects (data fusion).	[4], [10], [12], [17], [24], [25
Certain tools need to be selected for certain defect types.	[5], [6], [11], [13], [21]
Tools are to be continuously improved based on false negatives and false positives.	[4], [8], [10], [12]
Various machine learning techniques can help reduce false positive and negative defects.	[9], [14], [21],
Tools' usefulness can be improved by automatically suggesting solutions to some warnings.	[18], [20], [21]
Tools are configurable to fix warning messages and unwanted checks.	[5], [11]
Writing code in a way that reduces code complexity, coupling and global variables can help reduce false positive and negative defects.	[9], [12]
Ranking the bugs by weight or priority can improve the detection of actual bugs.	[17], [24]
Tools should only be used to compliment other review and testing techniques.	[23], [25]
Static code analysis tools can be integrated into frameworks for evaluating the tools.	[12]

Table 5 presents the solutions to disadvantages to answer RQ4. The solutions are amassed more loosely than previous tables as the suggestions are worded very differently yet they still share a general idea.

IV. DISCUSSION

A. RQ1 – Advantages

Table 2 has presented us with an overview of the most common static code analysis tool advantages. Many publications found it essential to bring out benefits like early detection of bugs, understanding the behaviour of the code and the reduced time-consumption aspect compared to manual static code review. There were also multiple mentions for that the tools can help solve more advanced problems that are difficult to see and fix instantly. The use of tools will help keep the coding standard high with the use of many best practices and keep the code review effort and noise (comments) to a minimum.

There are single mentions for the tools' high customizability, ease of achieving full code coverage, wide range of tools and general ease of use.

B. RQ2 – Disadvantages

Table 3 has presented us with an overview of the most mentioned issues with static code analysis tools. The most important problem was that the tools produce false positives for which the user cannot do anything. Also, a major disadvantage seemed to be false negatives, in which almost always, human involvement is needed.

In some cases, the tools cannot be used separately as a single tool cannot fix all the occurring problems. The tools are also mentioned to struggle with complex problems and complex control flow and coding techniques. There is concern for the quality of the warning messages and their compatibility across different tools. This also raised a

problem in a few publications that the tools' warning messages are not useful, and developers rarely even fix most of them.

Some publications found it problematic to even integrate and/or use the tools which contradicts the ease of use mentioned in the advantages. There are also mentions for that all the programming languages are not supported, high memory consumptions for some tools and low customizability for some efficient tools.

C. RQ3 – Tools

Table 3 has given us an overview of the most popular tools in our publications. FindBugs, PMD, Cppcheck and SonarQube were the tools that were most discussed. There are also multiple mentions for CheckStyle, Splint, Flawfinder, Coverty, FindSecBugs, Parasoft, Sparse and Clang.

Given that there were many single mentions for some tools across different programming languages, exclusion criteria could have filtered this paper to only focus on a single programming language. The tools mentioned once are currently therefore less meaningful. But the popular tools are indeed found.

D. RQ4 – Solutions

Table 4 has given us many solutions to various disadvantages to discuss RQ4. A most popular approach to solve many disadvantages would be to first choose the correct tool that has a good accuracy to solve the problems most common with the program. If then still problems persist, the tool should be combined with other tools for these tools to complement each other.

A more long-term solution is to report false negatives and positives as the tools are continuously improving. Machine learning can also help with reducing the false negatives and positives. Random Forest Technique was mentioned to have the best results [9].

The tools' configurability and customizability can also fix some of the false positives instantly and reduce unwanted checks. Another proactive solution is to write code in a way that reduces code complexity, coupling and global variables. These are cases in which some tools struggle to avoid false positive and negative cases. If the tools would rank the bugs by weight or priority, it would also be more helpful to the user to solve only the important issues.

V. Conclusion

There are many valid advantages and disadvantages to static code analysis tools. Yet one does not outweigh the other. RQ1 found that many benefits of the tools are undeniable and have huge benefits for developers and code reviewers, like tools helping with early detection and being less time-consuming than manual detection. RQ2 found all the most important issues with the use of the tools that one should be aware when using the tools, high false positive results and unclear warning messages being just a few. There are many tools to choose from (RQ3), FindBugs, PMD and CppCheck being the most mentioned, and they all have their own strengths. Considering the most popular solutions to the tools' issues (RQ4), many of the mentioned problems can easily be solved and there are also some workarounds in the form of combining different tools and tools continuously improving.

VI. References

[1] ISO/IEC/IEEE International Standard for Software Engineering - Software Life Cycle Processes - Maintenance, ISO/IEC 14764:2006 (E) IEEE Std 14764-2006 Revision of IEEE Std 1219-1998, pp. 1-58, 2006. Available from: https://doi.org/10.1109/IEEESTD.2006.235774.

[2] Nasir, Z., Abbasi, A., Z., "A framework for software maintenance and support phase", International Conference on Information and Emerging Technologies, Karachi, 2010, pp. 1-6. Available from: https://doi.org/10.1109/ICIET.2010.5625668.

[3] Gomes, I., Morgado, P., Gomes, T., Moreira, R., "An overview on the Static Code Analysis approach in Software Development", International conference on Computer Assisted Assessment, Zeist, 2014, pp. 1-3. Available from: https://doi.org/10.1007/978-3-319-08657-6_10.

[4] Stefanović, D., Nikolić, D., Dakić, D., Spasojević, I., Ristić, S., "Static Code Analysis Tools: A Systematic Literature Review", *Annals of DAAAM & Proceedings*, vol. 7, no. 1, pp. 565–573, Aug. 2020. Available from: https://doi.org/10.2507/31st.daaam.proceedings.078.

[5] Shaukat, R., Shahoor, A., Urooj, A., "Probing into code analysis tools: A comparison of C# supporting static code analyzers", *15th International Bhurban Conference on Applied Sciences and Technology (IBCAST)*, Bhurban, 2018, pp. 455–464. Available from: https://doi.org/10.1109/IBCAST.2018.8312264.

[6] Arusoaie, A., Ciobaca, S., Craciun, V., Gavrilut, D., & Lucanu, D., "A Comparison of Open-Source Static Analysis Tools for Vulnerability Detection in C/C++ Code", *19th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, Timisoara, 2017, pp. 161–168. Available from: https://doi.org/10.1109/SYNASC.2017.00035.

[7] Zampetti, F., Scalabrino, S., Oliveto, R., Canfora, G., Di Penta, M., "How Open Source Projects Use Static Code Analysis Tools in Continuous Integration Pipelines", *IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, Buenos Aires, 2017, pp. 334–344. Available from: https://doi.org/10.1109/MSR.2017.2.

[8] Maskur, A. F., Dwi Wardhana Asnar, Y., "Static Code Analysis Tools with the Taint Analysis Method for Detecting Web Application Vulnerability", *2019 International Conference on Data and Software Engineering (ICoDSE)*, Pontianak, 2019, pp. 1-6. Available from: https://doi.org/10.1109/ICoDSE48700.2019.9092614.

[9] Alikhashashneh, E. A., Raje, R. R., Hill, J. H., "Using Machine Learning Techniques to Classify and Predict Static Code Analysis Tool Warnings", *IEEE/ACS 15th International Conference on Computer Systems and Applications (AICCSA)*, Aqaba, 2018, pp. 1-8. Available from: https://doi.org/10.1109/AICCSA.2018.8612819.

[10] Fatima, A., Bibi, S., Hanif, R., "Comparative study on static code analysis tools for C/C++", *15th International Bhurban Conference on Applied Sciences and Technology (IBCAST)*, Bhurban, 2018, pp. 465–469. Available from: https://doi.org/10.1109/IBCAST.2018.8312265.

[11] Singh, D., Sekar, V. R., Stolee, K. T., Johnson, B., "Evaluating how static analysis tools can reduce code review effort", *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, Raleigh, NC, 2017, pp. 101-105. Available from: https://doi.org/10.1109/VLHCC.2017.8103456.

[12] Reynolds, Z. P., Jayanth, A. B., Koc, U., Porter, A. A., Raje, R. R., Hill, J. H., "Identifying and Documenting False Positive Patterns Generated by Static Code Analysis Tools", *IEEE/ACM 4th International Workshop on Software Engineering Research and Industrial Practice (SER&IP)*, Buenos Aires, 2017, pp. 55-61. Available from: https://doi.org/10.1109/SER-IP.2017..20.

[13] Zhang, S., Zhu, J., Liu, A., Wang, W., Guo, C., Xu, J., "A Novel Memory Leak Classification for Evaluating the Applicability of Static Analysis Tools", *IEEE International Conference on Progress in Informatics and Computing (PIC)*, Suzhou, 2018, pp. 351–356. Available from: https://doi.org/10.1109/PIC.2018.8706142.

[14] Flynn, L., Snavely, W., Svoboda, D., VanHoudnos, N., Qin, R., Burns, J., Zubrow, D., Stoddard, R., Marce-Santurio, G., "Prioritizing Alerts from Multiple Static Analysis Tools, Using Classification Models", *IEEE/ACM 1st International Workshop on Software Qualities and*

Their Dependencies (SQUADE), Gothenburg, 2018, pp. 30-20. Available from: https://doi.org/10.1145/3194095.3194100.

[15] Srisopha, K., Alfayez, R., "Software Quality through the Eyes of the End-User and Static Analysis Tools: A Study on Android OSS Applications", *IEEE/ACM 1st International Workshop on Software Qualities and Their Dependencies (SQUADE)*, Gothenburg, 2018, pp. 1–4. Available from: https://doi.org/10.1145/3194095.3194096.

[16] Krishnamurthy, R., Heinze, T. S., Haupt, C., Schreiber, A., Meinel, M., "Scientific Developers v/s Static Analysis Tools: Vision and Position Paper", *IEEE/ACM 12th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, Montreal, QC, 2019, pp. 89–90. Available from: https://doi.org/10.1109/CHASE.2019.00029.

[17] Dang, B. H., "A Practical Approach for Ranking Software Warnings from Multiple Static Code Analysis Reports", *SoutheastCon*, Raleigh, NC, 2020, pp. 1–7. Available from: https://doi.org/10.1109/SoutheastCon44009.2020.9368277.

[18] Marcilio, D., Furia, C. A., Bonifacio, R., Pinto, G., "Automatically Generating Fix Suggestions in Response to Static Code Analysis Warnings", *19th International Working Conference on Source Code Analysis and Manipulation (SCAM)*, Cleveland, OH, 2019, pp. 34–44. Available from: https://doi.org/10.1109/SCAM.2019.00013.

[19] Yang, J., Barrientes, C., Sanchez, J., Kim, Y. R., "Source Code Analysis for Secure Programming Practices", *International Conference on Computational Science and Computational Intelligence (CSCI)*, Las Vegas, NV, 2018, pp. 819–824. Available from: https://doi.org/10.1109/CSCI46756.2018.00164.

[20] Imtiaz, N., Rahman, A., Farhana, E., Williams, L., "Challenges with Responding to Static Analysis Tool Alerts", *IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, Montreal, QB, 2019, pp. 245–249. Available from: https://doi.org/10.1109/MSR.2019.00049,

[21] Koc, U., Wei, S., Foster, J. S., Carpuat, M., Porter, A. A., "An Empirical Assessment of Machine Learning Approaches for Triaging Reports of a Java Static Analysis Tool", *12th IEEE Conference on Software Testing, Validation and Verification (ICST)*, Xi'an, 2019, pp. 288–299. Available from: https://doi.org/10.1109/ICST.2019.00036.

[22] Smith, J., Johnson, B., Murphy-Hill, E., Chu, B., Lipford, H. R., "How Developers Diagnose Potential Security Vulnerabilities with a Static Analysis Tool", *IEEE Transactions on Software Engineering*, vol 45, no. 9, pp. 877-897, Sept. 2019. Available from: https://doi.org/10.1109/TSE.2018.2810116.

[23] Anderson, P., "The use and limitations of static-analysis tools to improve software quality", *CrossTalk*, vol. 21, no. 6, pp. 18–21, June 2008. Available from: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.170.5018&rep=rep1&type=pdf [Accessed 2nd April 2021].

[24] Mendonca, V.R., Rodrigues, C.L., Soares, F., Vincenzi, A.M., "Static Analysis Techniques and Tools: A Systematic Mapping Study", *The Eighth International Conference on Software Engineering Advances (ICSEA)*, Venice, 2013, pp. 1-15. Available from: https://doi.org/10.1016/j.infsof.2011.06.003.

[25] Lenarduzzi, V., Lujan, S., Saarimaki, N., Palomba, F., "A Critical Comparison on Six Static Analysis Tools: Detection, Agreement, and Precision", 2021. Available from: https://www.researchgate.net/publication/348739709_A_Critical_Comparison_on_Six_Static_Analysis_Tools_Detection_Agreement_and_Precision [Accessed 2nd April 2021]

[26] Novak, J., Krajnc, A., Žontar, R., "Taxonomy of static code analysis tools", *The 33rd International Convention MIPRO*, Opatia, 2010, pp. 418–422. Available from: https://www.researchgate.net/publication/251940397_Taxonomy_of_static_code_analysis_tools [Accessed 2nd April 2021]