Analysis of the Tools for Static Code Analysis

Danilo Nikolić, Darko Stefanović, Dušanka Dakić, Srđan Sladojević, Sonja Ristić
University of Novi Sad, Faculty of Techincal Sciences
Novi Sad, Serbia
E-mail: nikolic.danilo@uns.ac.rs

2021 20th International Symposium INFOTEH-JAHORINA (INFOTEH) | 978-1-7281-8229-2/20/$31.00 ©2021 IEEE | DOI: 10.1109/INFOTEH51037.2021.9400688

Abstract—**Static code analysis tools are being increasingly used to improve code quality. Such tools can statically analyze the code to find bugs, security vulnerabilities, security spots, duplications, and code smell. Various software tools are being developed to support developers in conducting static code analysis. In this paper, three tools to support static code analysis were analyzed and evaluated using the DESMET methodology. The tools were selected by conducting a systematic literature review in the field of static code analysis.**

Keywords-static code analysis; tools; evaluation; DESMET methodology;

I. INTRODUCTION

The source code's quality is a key factor in any software product and requires constant verification and monitoring. Static analysis is used to maintain and improve the quality of the source code. Static analysis of program code has been used since the early 1960s to optimize the operation of compilers [1]. Later, it proved useful for debugging tools, as well as for software development frameworks. A growing number of tools allow static code analysis, many of which allow an analysis of code written in different programming languages [2].

Static analysis tools are used to generate reports and point out certain deviations from the prescribed code quality standards. However, these tools do not allow automatic modifications to the source code. The decision to change the structure of previously written code is up to the software developers. Static code analysis tool helps software developers by, in addition to generating a report, stating the cause of a particular defect, as well as how this defect can be corrected.

These tools are increasingly finding application in industry, but one of the aspects of the application of these tools is certainly their inclusion in the education of future information technology engineers, as part of the e-learning system [3].

The aim of this paper is to compare the static code analysis tools most commonly used in research .

In order to select the tools to be analyzed, a systematic literature review in the field of static code analysis was conducted [4]. This literature review highlights tools in relation to the programming languages they support, according to the type of defects they detect, as well as the most commonly used tools for static code analysis in research. These tools were analyzed and evaluated using the DESMET methodology [5] in this paper.

In recent years, software tools are being developed in order to solve the current problems that developers are facing the quality of source code. In order to examine the usefulness of the developed tools in solving existing problems, it is necessary to evaluate these tools.

The remainder of the paper is organized as follows. In Section 2, the process of static code analysis is described, and the selected tools to be used in this paper are presented. In Section 3, the methodology for performing this study is presented. Results and discussion are presented in Section 4. Section 5 concludes the study and suggests future research.

II. STATIC CODE ANALYSIS AND SOFTWARE TOOLS

This section describes the process of static code analysis and selected tools.

A. Static code analysis

The process of static code analysis is useful not only for optimizing the compiler's operation (which was the original purpose) but also for detecting defects and possible shortcomings. In this way, it is possible to create tools that will help developers understand the program's behavior and identify various shortcomings of the program without its execution. The tools used for static code analysis are programs that explain the behavior of other programs [1].

The standard code review cycle includes four main phases [6]:

1. defining the goal,
2. launching a tool for static code analysis,
3. review of the source code (based on the obtained result of static analysis), and
4. source code reengineering.

In addition to the standard cycle, in Figure 1 several potential feedbacks or minor iterations between standard cycle steps are presented. This iterations are making the cycle more complex than the standard one. The number of iterations of this cycle and the need for feedback within the cycle depends of reengineering of code based on report generated by tool.

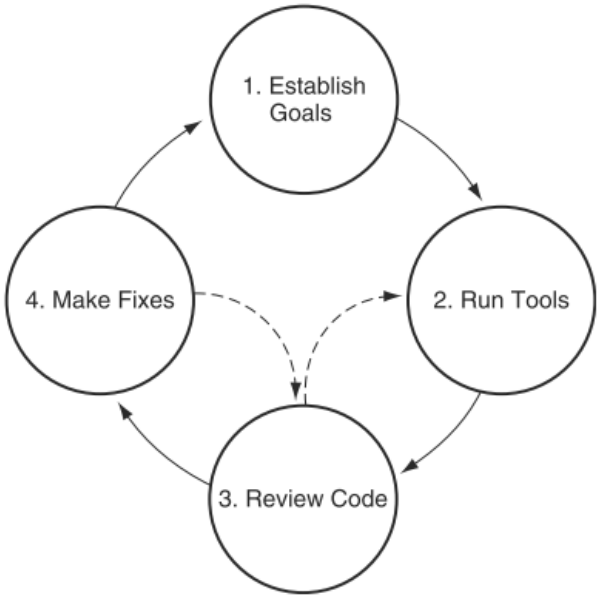

Figure 1. The code review cycle [6]

B. *Software tools for static code analysis*

Static code analysis tools look for a specific set of patterns or rules in the source code, very similar to the way antivirus programs detect viruses. More advanced tools allow practitioners to add new rules to a set of predefined ones. These tools cannot find an error if patterns or rules do not specify such behaviour. The most significant value of a static code analysis tool is the ability to identify common programming defects.
The static code analysis tools that are used in this research were selected based on a previously conducted systematic literature review described in detail in [4].
This review presents the static code analysis tools most commonly used in research and which tools are analyzed based on the type of defects they detect and the programming languages they support.
Six tools were presented, which were most often analyzed and presented in primary studies. These are tools that have been presented three or more times in different studies. In comparison, other tools presented in the studies covered by this review were analyzed in two or one studies. Presented tools are also used for conducting experiment on identification of strategies over static code analysis tools described in [7].
From six presented tools most often analyzed in primary studies, the first three are selected and used for this research.

III. METHODOLOGY

This section describes the methodology used to evaluate the three candidate tools, as well as the description of the chosen feature analysis approach

A. *DESMET methodology*

DESMET is a methodology for evaluating tools or methods. It was developed by Barbara Kitchenham [5] and consisted of nine evaluation methods suitable for different tool analyses. There are quantitive and qualitative methods in DESMET methodology [8]. In the guidelines, the first step to perform DESMET methodology is selecting an appropriate evaluation method. DESMET methodology is not flawless, so it has limits when trying to mix and match methods and tools as well as if there is no controllable development process in the organization [8]. Kitchenham defined evaluation criteria on the basis of which the DESMET methodologies are chosen.

B. *Feature analysis*

Feature analysis is a qualitative form of evaluation involving the subjective assessment of the relative importance of different features plus an assessment of how well each of the features is implemented by the candidate tools [8].
The objective of the feature analysis provides input for a decision whether to use or not use a tool in an organization. Evaluation should consider the following areas [8]:
1. suitability for purpose,
2. economic issues,
3. drawbacks, and
4. other advantages.

This methodology is also considered a subjective evaluation methodology.
Some of the criteria that should be considered when deciding whether to perform a feature analysis are [8]:

- a large number of methods/tools to assess and
- short timescales for evaluation exercise.

C. *Candidates and features*

This section provides an overview of the three candidate tools, the feature set against which the tools are evaluated and the approach taken to scoring the candidates.
The three candidate tools are:

1. Cppcheck – a open-source static analysis tool for C/C++ code. It provides a unique code analysis to detect bugs and focuses on detecting undefined behavior and dangerous coding constructs [9].
2. FindBugs – a program that uses static analysis to look for bugs in Java code. It is open-source software, distributed under the terms of the Lesser GNU Public License [10].
3. SonarQube – an open-source platform developed by SonarSource for continuous inspection of code quality. Sonar does static code analysis, which provides a detailed report of bugs, code smells, vulnerabilities, code duplications [11].

The features are divided into four sets relating to economics, ease of introduction, code review cycle support, and process management (Table 1). The following subsections describe the features in each of these sets.
The first two feature sets can apply to all tools, not just static code analysis tools, these feature sets are taken from feature sets presented in [12].

1) Feature Set 1: Economic

This set concerns economic factors relating to the initial cost of the tool and the subsequent support for maintaining (or upgrading) the tool. For this study, the highest scores are awarded if no initial payment is required (F1-SF01) and the tool is well (and freely) maintained by its developers, including having regular updates and a single point of contact for users to obtain support if needed (F1-SF02).

2) Feature Set 2: Ease of introduction and setup

This feature set focuses on the level of difficulty inherent in setting up and using the tool for the first time. Each tool should:

- have reasonable system requirements (F2-SF01) and not require any advanced hardware or software to function,
- have a simple installation and setup procedure (F2-SF02) that is supported by an installation guide (F2-SF03) or a tutorial (F2-SF04),
- be as self-contained as possible i.e. able to function, primarily, as a stand-alone application with minimal requirements for other external technologies (F2-SF05).

3) Feature Set 3: Code review cycle support

These features relate to how well the tool supports standard phases in the review cycle.

First, in the standard review cycle, the tool should provide an overview of all defects in the source code of the program (F3-SF01). The tool should also provide tracking of corrected defects in the source code reengineering process (F3-SF02).

In the review code phase, of the standard cycle, the tool should provide an indication of where the defect is in the source code (F3-SF03). Although static code analysis tools should not eliminate detected defects in the source code [6], only mark them, these tools should suggest a solution to eliminating the observed defect (F3-SF04). Reports generated by static code analysis tools should indicate the type of defect detected, as well as the level (for example critical/non-critical and major/minor defect) of the urgency of correcting the detected defect (F3-SF05). Finally, in this feature set, the tools should also indicate the estimated time to correct each defect in the source code, of course, this time may not always be accurate, but it should represent an estimate of the tool against the size of the detected defect (F3-SF06).

4) Feature Set 4: Process Management

This set of features relates to the management of a static code analysis process. Static code analysis could be a collaborative process. Therefore, the tool should allow multiple users to work on a single project (F4-SF01). It should support document management (F4-SF02), in particular, managing large projects. The tool should be secure (F4-SF03) and include a user log-in or similar system. It should be able to manage the roles of users (F4-SF04). For example, it would be useful to state which user will eliminate which types of defects. Finally, the tool should be able to support multiple static code analysis projects (F4-SF05).

The list of features is shown in Table 1.

D. Scoring Candidate tools

Three elements of scoring process are:

- scoring each tool against each feature to produce a raw score,
- assigning a level of importance to each feature which is used as a weighting (i.e. a multiplier) to convert raw scores to weighted scores for each feature, and
- determining scores for each feature set and an overall score for each candidate tool.

A detailed explanation of the scoring can be seen in [12].

TABLE I – FEATURES USED IN ANALYSIS

id	Feature Set	id	Subfeature	Subfeature Level of Importance	Interpretation of Judgment Scale	Feature Set Importance Weighting
F1	Economic	F1-SF01	The tool does require financal payment to use	HD	JI1	0.1
		F1-SF02	Maintenance	HD	JI1	
F2	Ease of introduction and setup	F2-SF01	The tool has reasonable system requirements.	M	JI1	0.2
		F2-SF02	Simple installation and setup	HD	JI2	
		F2-SF03	There is an installation guide	HD	JI1	
		F2-SF04	There is a tutrial	HD	JI1	
		F2-SF05	The tool is self-contained	HD	JI1	
F3	Code review cycle support	F3-SF01	Overview of defects in the source code	D	JI1	0.4
		F3-SF02	Tracking of corrected defects	HD	JI3	
		F3-SF03	Location of defects	HD	JI3	
		F3-SF04	Suggesting a solution	HD	JI3	
		F3-SF05	Indicating type of defects	N	JI1	
		F3-SF06	Indicating estimated time to correct defect	N	JI1	
F4	Process Management	F4-SF01	Support for multiple users	M	JI1	0.3
		F4-SF02	Document management	D	JI1	
		F4-SF03	Security	D	JI1	
		F4-SF04	Management of roles	HD	JI1	
		F4-SF05	Support for multiple projects	M	JI1	

Subfeature levels of importance are [12]:

1. Mandatory feature – M – 4 points
2. Highly Desirable – HD – 3 points
3. Desirable – D – 2 points
4. Nice to have – N – 1 point

Interpretations of the Judgement Scales are [12]:

1. J11 – Is the feature present?
2. J12 – Is the tool simple to install and setup?
3. J13 – Is the activity supported?

IV. RESULTS

This section provides an overview of the scoring for each candidate tool. The results for all of the candidate tools are summarized in Table 5.

A. Results for Cppcheck

In this subsection, the results of the candidate tool Cppcheck are presented. Table 2 presents the results of this tool.

1) Feature Set 1

Cppcheck tool is free to use and can be accessed from the development team's website. The tool is well maintained, regularly updated (the last update was in December 2020), and provides a single point of contact for a user to obtain help if needed. Cppcheck scored 6 of 6 (6/6) in this feature set.

2) Feature Set 2

Cppcheck can be downloaded from the developers' site, also, it can be installed and started locally. Cppcheck has a partly complex setup. Installation instructions can be found on the tool's website. Currently, there is no tutorial. Cppcheck scored 8 out of 16 (8/16) marks for this feature set.

3) Feature Set 3

In terms of code review cycle support, the Cppcheck tool has provided an overview of all defects in the program's source code when generating reports. Tracking corrected defects in the project is only possible by restarting the tool over the source code. However, this tool shows where the defect is in the source code and suggests possible solutions for the detected defect. Cppcheck indicates defects with the CVE (Common Vulnerabilities and Exposures) [13] type designation, however, it does not indicate the estimated level of severity of the defect as well as the time required for its elimination. This tool scored 8/13 in feature set 3.

4) Feature Set 4

Cppcheck allows multiple users but this mode does not provide support for tracking updates. This tool allows multiple projects to be undertaken. The tool contains a number of useful document management features. Cppcheck has no role support provided. Cppcheck implements a login system, which requires a username and password. This tool scored 10/16 in this feature set.

TABLE II – SCORES FOR CPPCHECK

Feature Set	Sub Feature	Weighted Score	Feature Set Score	% Feature Set Score
F1	F1-SF01	3	6/6	100%
	F1-SF02	3		
F2	F2-SF01	2	9.5/16	59.38%
	F2-SF02	3		
	F2-SF03	3		
	F2-SF04	0		
	F2-SF05	1.5		
F3	F3-SF01	2	8/13	61.54%
	F3-SF02	0		
	F3-SF03	3		
	F3-SF04	3		
	F3-SF05	0		
	F3-SF06	0		
F4	F4-SF01	2	10/16	62.5%
	F4-SF02	2		
	F4-SF03	2		
	F4-SF04	0		
	F4-SF05	4		
Total Score			**Overall % Score Using Feature Set**	
33.5/51			**65.69%**	

B. Results for FindBugs

In this subsection, the results of the candidate tool FindBugs are presented. Table 3 presents the results of this tool.

1) Feature Set 1

FindBugs tool is free to use and can be accessed from the development team's website. The tool is partly maintained, not updated since March 2015. and provides a single point of contact for a user to obtain help if needed. FindBugs scored 3 of 6 (3/6) in this feature set.

2) Feature Set 2

FindBugs can be downloaded from the developers' site, also, it can be installed, started locally or added in IDE (Integrated Development Environment). FindBugs has a partly simple setup. Installation instructions can be found on the tool's website. There is no tutorial for this tool. FindBugs scored 8 out of 16 (8/16) marks for this feature set.

3) Feature Set 3

In terms of code review cycle support, the FindBugs tool has provided an overview of all defects in the program's source code when generating reports. Tracking corrected defects in the project is not possible in the single start of the tool. This tool report where the defect is in the source code (in a defect report, not directly in code). FindBugs indicates defects, and the estimated level of severity of the defect, however, it does not estimate the time required for its elimination. This tool scored 6/13 in feature set 3.

4) Feature Set 4

FindBugs does not allow multiple users. This tool allows multiple projects to be undertaken. The tool can be added to IDE, which makes it easier to track defects in the source code.

FindBugs has role support provided. This tool implements a login system. FindBugs scored 10/16 in this feature set.

TABLE III – SCORES FOR FINDBUGS

Feature Set	Sub Feature	Weighted Score	Feature Set Score	% Feature Set Score
F1	F1-SF01	3	3/6	50%
	F1-SF02	0		
F2	F2-SF01	2	8/16	50%
	F2-SF02	3		
	F2-SF03	3		
	F2-SF04	0		
	F2-SF05	0		
F3	F3-SF01	2	6/13	46.15%
	F3-SF02	0		
	F3-SF03	1.5		
	F3-SF04	1.5		
	F3-SF05	1		
	F3-SF06	0		
F4	F4-SF01	0	11/16	68.75%
	F4-SF02	2		
	F4-SF03	2		
	F4-SF04	3		
	F4-SF05	4		
Total Score			**Overall % Score Using Feature Set**	
28/51			**54.9%**	

C. Results for SonarQube

In this subsection, the results of the candidate tool SonarQube are presented. Table 4 presents the results of this tool.

1) Feature Set 1

SonarQube community version is free to use and can be accessed from the development team's website. However, not all functionalities of this tool are available in the community version, but only some. The tool is well maintained, regularly updated (the last update was in December 2020),, and provides a single point of contact for a user to obtain help if needed. SonarQube scored 4.5 of 6 (4.5/6) in this feature set.

2) Feature Set 2

SonarQube can be downloaded from the developers' site, also, it can be installed and started locally. SonarQube has a complex setup. To configure a full version of SonarQube, a number of external technologies must also be installed depending on the programming language to be analyzed. SonarQube scored 8 out of 16 (8/16) marks for this feature set.

3) Feature Set 3

When it comes to supporting the code review cycle, the SonarQube tool has provided an overview of all defects in the program's source code when generating reports. Tracking corrected defects in the project is only possible by restarting the tool over the source code. However, this tool shows where the defect is in the source code and suggests possible solutions for the detected defect. SonarQube for each defect indicates a certain level of severity and size of the defect, as well as the estimated time required to correct the defect. This tool scored 10/13 in feature set 3.

4) Feature Set 4

SonarQube allows multiple users but this mode does not provide support for tracking updates. This tool allows multiple projects to be undertaken. The tool contains a number of useful document management features. The tool can be added to IDE, which makes it easier to track defects in the source code. SonarQube has role support provided. This tool implements a secure login system, which requires a username and password on each visit. SonarQube scored 13/16 in this feature set.

TABLE IV – SCORES FOR SONARQUBE

Feature Set	Sub Feature	Weighted Score	Feature Set Score	% Feature Set Score
F1	F1-SF01	1.5	4.5/6	75%
	F1-SF02	3		
F2	F2-SF01	2	8/16	50%
	F2-SF02	1.5		
	F2-SF03	3		
	F2-SF04	0		
	F2-SF05	1.5		
F3	F3-SF01	2	10/13	76.92%
	F3-SF02	0		
	F3-SF03	3		
	F3-SF04	3		
	F3-SF05	1		
	F3-SF06	1		
F4	F4-SF01	2	13/16	81.25%
	F4-SF02	2		
	F4-SF03	2		
	F4-SF04	3		
	F4-SF05	4		
Total Score			**Overall % Score Using Feature Set**	
35.5/51			**69.61%**	

D. Discussion

This section presents the discussion of the results and limitations of the study.

DESMET methodology is a subjective method of evaluation and the presented results were obtained by the author's use and analysis of the use of these tools on different projects. Table 5 presents the summary results for all three tools used in this research.

TABLE V – FEATURE SET SCORES

TOOL	F1	F2	F3	F4	Total
Cppcheck	100%	59.4%	61.5%	62.5%	**65.6%**
FindBugs	50%	50%	46.1%	68.7%	**54.9%**
SonarQube	75%	50%	76.9%	81.2%	**69.6%**

SonarQube achieved the best results in this analysis, 69.61% of the total score. As the differences in the percentages of other tools are not overly different, what SonarQube pointed out as better than other tools is better support for code review cycle and F4 – Process Management. The tools show similar results, indicating that the authors chose tools with relatively similar capabilities and options for developers. Comparing SonarQube to two other tools that are evaluated in this study, one of the biggest disadvantages is in F1. SonarQube is free only in the

community version, while Cppcheck and FindBugs are open sources for the full version. The disadvantages of all tools are reflected when looking at F2, none of the presented tools contains an official tutorial describing the setup and how to use the tool. Also, no tool allows bug fixes tracking without restarting the tool.

V. CONCLUSION

In this paper, the analysis and evaluation of three tools for static code analysis are presented. DESMET methodology in the paper was used. Four sets of features are described, on the basis of which the presented tools for static code analysis are evaluated and analyzed.

Static code analysis tools are increasingly finding their application and use in the work of software developers in different positions. New tools are being developed, and existing tools are being improved every day, offering their customers an increasingly high-quality service by creating confidence in the results of static analysis and enabling them to avoid certain code defects and make their software solution better.

Further development of tools for static code analysis implies their inclusion in development environments in the software development process, with the ultimate goal of creating digital assistants that should support developers in the process of developing quality software solutions.

Further research in this field should include a comparison of the results and reports generated by these tools in terms of the types of defects they recognize.

ACKNOWLEDGEMET

This research has been supported by the Ministry of Education, Science and Technological Development through the project no. 451-03-9/2021-14/200156: “Innovative scientific and artistic research from the FTS domain”.

REFERENCES

[1] A. Moller and I. Schwartzbach, "Static program analysis," 2019.

[2] M. Beller, R. Bholanath and M. S., "Analyzing the state of static analysis: A large-scale evaluation in open source software," 2016.

[3] T. Lolić, S. Ristić, D. Stefanović i U. Marjanović, „Acceptance of E-Learning System at Faculty of Technical Sciences,“ u Central European Conference on Information and Intelligent Sytems, Varaždin, 2018.

[4] D. Stefanović, D. Nikolić, D. Dakić, I. Spasojević i S. Ristić, „Static code analysis tools: A systematic Literature Review,“ u Proceeding of of 31st DAAAM International Symposium, Vienna, 2020.

[5] B. Kitchenham i S. Linkman, „DESMET: A method for evaluating Software Engineering methods and tools.,“ 2000.

[6] J. West i B. Chess, „Secure programming with static code analysis,“ Pearson education, 2007.

[7] D. Stefanović, D. Nikolić, S. Havzi, T. Lolić i D. Dakić, „Identification of strategies over tools for static code analysis“ (in press) u Proceeding of the 9th International Conference on Engineering and Technology, Krabi, 2021.

[8] B. Kitchenham, S. Linkman i D. Law, „A methodology for evaluating software engineering methods and tools,“ Lect. Notes Comput. Sci., 1993, pp. 121-124.

[9] C. Chandrapal , C. S. Vishal i L. D. Manik, „Code Analysis for Software and System Security Using Open Source Tools,“ Information Security Journal, t. 21, br. 6, pp. 346-352, 2012.

[10] F. Zampetti, S. Scalabrino, R. Oliveto, G. Canfora i M. Di Penta, „How Open Source Projects use Static Code,“ 2017.

[11] G.-M. Javier, G.-V. Marisol i E.-B. Julio, „Improved Metrics Handling in SonarQube for Software Quality Monitoring,“ u Advances in Intelligent Systems and Computing, 2016.

[12] C. Marshall, P. Brereton i B. Kitchenham, „Tools to Support Systematic Reviews in Software Engineering: A Feature Analysis,“ u ACM Int. Conf. Proceeding Ser., 2014.

[13] „CVE - Common Vulnerabilities and Exposures,“ The MITRE Corporation, 12 Januar 2021. [Na mreži]. Available: https://cve.mitre.org