A Comparative Analysis of Static Code Analysis Tools that check Java Code Adherence to Java Coding Standards

Conference Paper · March 2019

DOI: 10.1109/C-CODE.2019.8681007

CITATIONS
5

READS
692

3 authors, including:

Qirat Ashfaq
NUST
2 PUBLICATIONS **7** CITATIONS
SEE PROFILE

Rimsha Khan
National University of Sciences and Technology
4 PUBLICATIONS **8** CITATIONS
SEE PROFILE

A Comparative Analysis of Static Code Analysis Tools that check Java Code Adherence to Java Coding Standards

Qirat Ashfaq
Department of Software Engineering
NUST College of Electrical and Mechanical Engineering,
Rawalpindi, Pakistan
qiratashfaq76@gmail.com

Rimsha Khan
Department of Software Engineering
NUST College of Electrical and Mechanical Engineering,
Rawalpindi, Pakistan
rimshakhanrk777@gmail.com

Sehrish Farooq
Department of Software Engineering
Fatima Jinnah Women University,
Rawalpindi, Pakistan
sehrishfarooq01@gmail.com

***Abstract*—Java programming language is considered highly important due to its extensive use in the development of web, desktop as well as handheld devices applications. Implementing Java Coding standards on Java code has great importance as it creates consistency and as a result better development and maintenance. Finding bugs and standard's violations is important at an early stage of software development than at a later stage when the change becomes impossible or too expensive. In the paper, some tools and research work done on Coding Standard Analyzers is reviewed. These tools are categorized based on the type of rules they checked, namely: style, concurrency, exceptions, and quality, security, dependency and general methods of static code analysis. Finally, list of Java Coding Standards Enforcing Tools are analyzed against certain predefined parameters that are limited by the scope of research paper under study. This review will provide the basis for selecting a static code analysis tool that enforce International Java Coding Standards such as the Rule of Ten and the JPL Coding Standards. Such tools have great importance especially in the development of mission/safety critical system. This work can be very useful for developers in selecting a good tool for java code analysis, according to their requirements.**

Keywords— analyzers; static analysis; standards; critical system; testing; style; exceptions; quality; security.

I. INTRODUCTION

In various applications, Java is being extensively used e.g. in games, enterprise software. As Java is used in many areas where different aspects of software quality such as reliability, software safety etc. play an important role, therefore validation of Java code becomes very important. If there are errors in code then at later stages it would be hard to detect and correct those errors, therefore it is necessary to find bugs in early stages. Static code analysis help developers identify potential bugs by indicating them before the execution of code. Hence, good validation tools are required to detect potential bugs and also to check coding standards against rules.

Nowadays, there are validation tools that are being used to detect errors in program. Different validation tools use different techniques such as data flow analysis, control flow analysis, syntactic pattern matching, theorem proving, model checking etc.

In this paper, some important Java code analyzing tools are described. Some of these tools help to find and detect potential bugs and some of them check coding standards and metrics. The purpose of discussing and analyzing these tools is to help java developers select a tool according to their need as well as help researchers to design a new tool that will check and meet coding standards against defined rules.

The rest of the paper is organized as follows. Section II presents literature survey. Section III and IV contain detailed analysis of different tools for Java language. In Section IV, tools are compared against some selected attributes. The paper is concluded at Section V.

II. LITERATURE REVIEW

In static code analysis, the source code is analyzed for conformance to coding standards without executing the program. In a study [9], a comparison has been carried out among open source tools for Java, including: Squale, Sonar, EvalMetrics. JSC is Java static checker discussed in the paper [2]. It is a Java code analyzer tool. JSC is Java static checker discussed in [2]. JSC uses standard compiler optimization algorithms. PMD, JLint, FindBugs, JSC are open source tools for Java [4]. These tools check general errors in Java code. The referred paper compares selected tools and concludes that FindBug is most commonly used tool for potential bugs. Some commonly used static code analyzers are presented in [6]. The authors have tried to classify some tools with respect to supporting programming languages and their capabilities. The paper compares and analyzes different coding standards for Java and the tools that check conformance to these standards [11].

An important aspect of code analyzers is detecting exceptions. Exceptions handling related tools are analyzed in the paper [1]. For developing secure software, it is essential that code must be error free. There are some tools that remove exceptions from code and ensure security. In mutation testing technique some configuration of program is done at small level [3]. Two tools, Pitest and μJava, are analyzed for handling exceptions in code. In these, exceptions are introduced in code and then it is checked which tool is fastest in performance for handling exceptions. Some of the papers that were analyzed used a metric based approach towards analysis.

Metrics and methods are used to evaluate the effectiveness of static analyzers [5]. In this work, different but important features of assessing tools are explained. An overview of open source software tools from the point of view of static analysis of Java source code that automate the gathering of Internal Quality metrics for Software Products, is presented in [8]. Sixteen tools are analyzed in the referred paper. Some Major organization have also presented their research and analysis on code analyzers. A code analyzer tool designed especially for the large Google Java code base and built over a third-party compiler is analyzed in the paper [10].

Three main tools discussed in the paper are Thindex, Java dependency and error prone. A research on 30 different analysis

978-1-5386-9609-5/19/$31.00 ©2019 IEEE

tools was carried out at CERN [12]. Their work mainly focused on security tools and came up with a list of efficient tools that can greatly help developers chose a good tool according to their requirements. Three static analysis tools are compared; CheckThread, RacerX and RELAY. NASA and IBM supplies open source benchmark programs and these three tools are used to find concurrency errors in that programs [7]. Similar work is done in a paper [17] in which FindBugs, PMD, Checkstyle, and UCDetector are evaluated and compared.

Table-I shows the parameters and the possible values. These parameters have been extracted and are limited by the scope of paper under study.

.

III. PARAMETER TABLE AND ANALYSIS TABLES

TABLE I. PARAMETER TABLE

Sr. No	Evaluation Parameters	Possible Values
1	Input	a. **Source code (S)** – source file can be loaded b. **Byte code (B)** – file having Microsoft Intermediate Language (MSIL) or Java Byte code can be loaded c. **Jar file (J)**- it is used to aggregate many Java class files and associated metadata and resources
2	Rules	a. **Style (s)** – checks the visualization of the source code b. **General (G)** –static code analysis's general rules and also contain rules related to naming c. **Concurrency (c)** –errors of duplicate code d. **Exceptions (E)** – errors that occur by throwing or not throwing exceptions e. **Quality (Q)**– errors deals with the performance, maintainability or other Quality parameters f. **Security (S)** – errors that show impact on security of the application g. **Dependency (D)**- allows to find unwanted dependencies between two sets of classes h. **Compatibility (C)**- check for compatibility with older versions
3	Extensibility	a. **Possible (P)**– it is possible to extend the tool b. **Not Possible (NP)** – it is not possible to extend the tool
4	Availability	a. **Open Source (OS)** –source code is available and tool is free b. **Free (F)** –source code is not available but tool is free c. **Commercial (C)** – payment required for tool
6	Reports	a. **Unparsable (UP)**–those tools that would require unreasonable effort to convert their reports into something that could be read and understood directly. b. **Parsable (P)**–those tools whose output was easy to convert into a clear and legible format. c. **Readable (R)**–those tools that had informative and readable output.
7	Latest Release Version/Date	**Different Latest Release Version and Date for different tools**
8	Errors	**Different type of errors for different tools**
9	Violation of Rules	a. **High (H):** shows whether risk is high b. **Medium (M) :** shows whether risk is normal c. **Low (L):** whether risk is low
10	Output	a. **Text file (TF)** – tool can present results in plain format, in text file. b. **List (L)**– tool can present results in in GUI c. **XML file (X)**– tool present results in XML d. **HTML file (H)**– tool can present results in HTML

TABLE II. ANALYSIS OF JAVA TOOLS

S. No	Tool	Input	Rules	Availability	Extensibility	Reports	Release Version/Date	Errors	Violation of Rules	Output
[1]	**Checkstyle**	Source code	General, Style,	Open source	Possible	Readable, XML file, List, HTML	7.5.1 / February 4, 2017	Naming conventions, length, line, white spaces errors	Medium	List, HTML, XML file
[2]	**CheckThread**	Source code	Concurrency	Open source	Possible	Readable, List	1.0.9.1/ April 29, 2009	Find threading bugs at compile time	Medium	List
[3]	**Ckjm**	Byte code	General	Open source	Possible	Parsable, XML	1.9 / April 4, 2012	calculates Chidamber and Kemerer object-oriented metrics	Medium	XML
[4]	**Classycle**	Byte code	Dependency	Open source	Possible	Parsable, XML	1.4 1/ Nov 1, 2014	Dependency errors	Medium	XML
[5]	**Clirr**	Jar file	Compatibility	Open source	Not Possible	Readable, XML	1.4-10/Feb 10, 2017	Java Compatibility issues	Medium	XML
[6]	**Condenser**	Jar file, Source code	Concurrency	Open source	Possible	Readable List	1.05	Finding and removing duplicated Java code.	Low	List
[7]	**Coqua**	Source Code	General, Style	Free	Possible	Readable, List	1.0.1	Management issues	Medium	List
[8]	**Crap4j**	Source Code	Quality	Open Source	Possible	Readable, List	1.1.6	Code difficult to test, understand, or maintain	High	List
[9]	**Dead Code Detector**	Byte code	Quality	Free	Possible	Readable, List	NetBeas IDE 7.3.1	Detect dead code	Low	List
[10]	**Dependency Finder**	Byte code	Dependency	Open source	Possible	Readable, Graph	1.2.1-beta4	Dependency errors through graphs and mining for information	Medium	Graph

[11]	**DoctorJ**	Source Code	General, Exceptions	Open source	Possible	Readable, Plain Text Format	5.1.2	Misspelled words, exception names and parameter.	High	Plain Text Format
[12]	**Emma**	Source, byte code	Quality	Open source	Not possible	Readable plain text, HTML, XML	2.1 / May 13, 2005	Measure, report Java code coverage and errors which effects performance.	High	List
[13]	**FindBugs**	Source code	Genera, Style	Open Source	Possible	Readable, XML	3.0.1 / March 6	Potential bugs and performance issues	High	List, XML
[14]	**Hammurapi**	Source, byte code	Quality	Open source	Possible	Readable, HTML	3.0.0	Potential problems	High	HTML
[15]	**Jalopy**	Source code	Quality	Open source	Not possible	Readable, plain text	1.9.4_108	Detect and fix number of code conventions	Low	List
[16]	**JarAnalyzer**	Byte code	Quality	Open source	Not possible	Readable, XML	N/A	Check .jar files	Low	Xml
[17]	**JavaNCSS**	Source code	Quality	Free	Not possible	Readable, plain text, XML	30.51 7/2/2009	Measures two standard code metrics	Low	List, Xml
[18]	**JCSC**	Source code	Style	Free	Possible	Parsable, HTML	1.5	Bad code, weaknesses and standards violation	High	HTML
[19]	**JDepend**	Byte code	Dependency	Open source	Possible	Readable, Graphical, List, XML	2.9	Extensibility, reusability, and maintainability issues	Medium	Graphical, List, XML
[20]	**Lint4j**	Byte code	Security, performance	Open Source	Not possible	Readable	0.9.1	Locking, performance and threading issues	High	List
[21]	**Macker**	Source, byte code	General	Open Source	Possible	Readable	0.4	Naming, access rules, filters etc.	Medium	List
[22]	**QJ-Pro**	Source code	General	Open source	Possible	N/A	Mar 22, 2005	Coding standards, code structure potential bugs, misuse of Java language	High	Charts
[23]	**Sonar Java**	Source code	Concurrency, General	Open Source	Possible	Parseable	4.13	Duplicates, standard violations, bugs complexity errors etc.	Medium	Lists, charts etc.
[24]	**Spoon**	Source, Byte code	General	Open source	Possible	Parseable	5.9/ September 6, 2017	Empty catch blocks, circular references between packages, factory pattern.	Medium	Auto fix

TABLE III. ANALYSIS OF MULIPLE LANGUAGE TOOLS

S. No	Code Quality Tools	Input	Rules	Availability	Extensibility	Reports	Release Version/Date	Errors	Violation	Output
[1]	**Axivion Bauhaus Suite**	Source code	Dependency	Closed source	Not possible	Readable	5	Dead code, cyclic dependencies, guidelines violation etc.	Medium	List
[2]	**PMD**	Source code	General, Style, Concurrency	Open source	Possible	Readable	5.8.1/ July 1, 2017	Potential problems, Dead code, possible bugs, duplicate code and overcomplicated expressions	High	List
[3]	**Squale**	Source code	Quality	Open source	Possible	Readable	7.1/May 26, 2011	Quality of your software developments	Medium	Auto fix
[4]	**CAST**	Source code	Accuracy, precision	Closed source	Not possible	Readable	December 7, 2010	Standards violation	High	List
[5]	**Checkmarx**	Source code	Security,	Open source	Possible	Readable	N/A	Vulnerabilities, rules violations	Medium	List
[6]	**Cigital SecureAssist**	Source code	Security	Closed source	Not possible	Readable	3.0.3/ June 2, 2017	Injections, improper error handling etc.	Medium	List
[7]	**Codacy**	Source code	Performance, security	Open source	Possible	Readable	2.0.1	Vulnerabilities, code duplication etc	Low	List, graphs
[8]	**Coverity**	Source code	General, Security	Open source	Not possible	Readable	N/A	Direct object reference, injections etc	Medium	List
[9]	**DefenseCodeThunderScan**	Source code	Security	Free Trail	Possible	Readable	2.0.8/ May 24, 2017	Security audits of application source code	High	List
[10]	**Facebook Infer**	Source code	General, concurrency, Exceptions	Open Source	Possible	Readable	0.12.1	Null pointer resource leaks,, annotation reachability, lock guards, concurrent race conditions	Medium	List
[11]	**GrammaTech**	Source code, binaries	Security, General	Commercial	Possible	Readable	3.6	Identifies security vulnerabilities and programming bugs in software	High	List
[12]	**HP Fortify**	Source code	Security	Commercial	Possible	Readable	N/A	security liability early in the software development lifecycle	Medium	List

[13]	**IBM Security**	Source code	Security	Commercial	Possible	Readable	9.3	Security vulnerabilities, by integrating test with development	High	List
[14]	**Kiuwan**	Source code	Security, General	Commercial	Possible	Readable	N/A	Defect detection, Application security , IT Risk Management	High	List
[15]	**Klocwork**	Source code	MISRA, ISO 26262, Security	Commercial	Possible	Readable	9.5/ Jan, 2012	Security vulnerability, standards compliance (MISRA, ISO 26262 and others), defect detection	High	List
[16]	**Dependometer**	Bytecode	Dependancy	Open source	Possible	Readable	1.18	Dependencies checks against logical architecture	Medium	Graph
[17]	**SonarQube**	Source code	security	Open source	Possible	Readable, html	6.5 / August 3, 2017	Inspection of code quality	High	HTML
[18]	**Yasca**	Source code	Security, Quality	Open source	Possible	Readable	2.2 / June 4, 2010	Security vulnerabilities, code quality, performance	Medium	HTML, XML

IV. Analysis

The papers were thoroughly analyzed and parameters were extracted that were measurable, diagnostic, definable and varying. Based on the parameters defined in Table I, the tools were evaluated in Table II and Table III. After analysis, the tools were categorized into 7 major categories based on the type of rules a tool enforces. The categories are: style, concurrency, exceptions, quality, security, dependency, compatibility and general methods of static code analysis. The tools are further described in the context of the coding rules that each of it analyzes.

A. Quality

Software Quality can be defined as the degree of conformance to explicit or implicit requirements and expectations. Maintaining software quality is very important especially for critical system. Squale [3], DCD (Dead Code Detector) [9], Hammurapi [14] are some of the tools that checks Java code against coding standards that can result in better quality software. Hammurapi maintains quality by looking for potential problems. DCD detects dead code in the project. Squale helps in enhancing quality of the software by meeting quality requirements with support for multi languages.

Quality of software can be enhanced by performance testing. Performance of software can be defined as the ability of software to function correctly even in a critical situation. It can be improved by implementing performance related rules. Yasca [18] is an open source tool that ensures good coding practiced being followed and checks for performance and vulnerabilities in the code. Lint4j [20] is also an open source tool that checks the quality of the software in terms of performance. It detects and reports issues like locking and threading issues. Checkstyle [1] and Codacy [7] checks performance rules violation and indicate errors like naming conventions, length, line, whitespaces, and also Java document that is improper or missed.

Maintainability is another important factor of quality of software. It is very important that software adapts to change in environment or other modification easily. There are some good tools that check the code with respect to rules of maintainability. Crap4j [8] ensures maintainability by identifying and reporting code that is difficult to test, understand or maintain. Emma [12] ensures maintainability through a code coverage tool that supports small fun projects for an individual person to a team. Jalopy [15] is another code analysis tool that enforces maintainability and detects code conventions and potential bugs in the project. Jar analyzer [16] is an open source tool that checks for .jar files and coding standards resulting in maintainable software.

Considerable amount of work is done by P. Tomas, M. J. Escalona, and M. Mejias on quality implementing Java Tools [8]. Their paper gives an overview of open source software tools from the point of view of static analysis of Java source code that automate the gathering of Internal Quality metrics for Software Products. Sixteen tools are analyzed in referred work. In another paper [9], a comparison has been carried out between three software Sonar, EvalMetrics and Squale are discussed.

B. Dependency

Dependency is how much a software module relies on another module. It is important to implement software with clean architecture where each of the modules has least dependencies and coupling. Modularization of code marks better reuse of the code and compilation time. It can be reduced by using tool like Classycle [4] which is an open source tool that detects cyclic dependencies between packages as well as classes and especially cyclic dependencies. Jdepend [19] is another important tool that besides checking cyclic dependencies like Classycle, also detects third party dependencies and invert dependencies hence maintaining software quality. Dependency Finder [10] creates dependency graphs from the code and then use them for receiving useful information. Axivion Bauhaus Suite [1] can detect different types of cyclic dependencies including call cycles, component cycles and include cycles.

Dependency tools were analyzed in a paper by Aftandilian, E. Sauciuc, R.Priya, and S.Krishnan [10].In the paper three tools were explained, the Strict Java dependency, the error prone and Thindex. Java dependency tool enforces Google's dependencies policies. The error prone can find common errors and API which help developers to know errors while developing. Finally, Thindex resolves indexing problems hence supporting large Google projects.

C. Concurrency

Concurrency occurs as a result of duplicate code also called clones. A duplicate code can be either totally identical to a code already present in a project or somewhat identical to it. Detecting and removing duplicate code results in lesser lines of code and as a result lesser compilation time. Sonar Java [23] detects duplicates, standard violations potential bugs and user defined rules. Condenser [6] detects and automatically removes duplicate code in a Java program. CheckThread [2] catches concurrency bugs at compile time. It also detects race condition which is an unwanted state in which two or more threads try to access same data. Facebook Infer [10] too detects race condition.

An important software that detects duplicate code is PMD [2]. It not only checks concurrency rules but also some other important rules. It was concluded in the paper [11] the best tool for static analysis is PMD.

Prominent work on static tools that analyzes dependency rules is done in [7]. The paper compares and analyzes three static analysis tools: CheckThread, RacerX and RELAY. NASA and IBM supplied open source benchmark programs are tested for concurrency bugs using these three tools. The analysis has been done to find how efficiently these tools find errors in open source programs or not. In Conclusion, if static tools are utilized in combinations then beneficial for finding spurious bugs.

D. Style

Style in programming is a set of guidelines or rules which are used while writing the source code. Programming style that is particular will be helpful for coder to better understand and read source code. Checkstyle [1] detects naming conventions errors, white space, line, wrong use of brackets and also Java documents that are misused. Coqua [7] measures five different Java code quality metrics, and provide an overview and history for the management. FindBugs [13] checks errors of bad practices as well as potential errors in code. PMD [2] finds potential bugs, possible bugs, Dead code, duplicate code and overcomplicated expressions.

JCSC [18] checks against coding standard and bad code, weakness in code. Different tools were analyzed in a [11] with respect to different standards. Style was also one of the parameter for comparative study. PMD along with other tools that check and apply Java standards, styles, and some other standards were compared and analyzed. 20 different Java Coding standards were compared and analyzed against certain parameters, including characteristics, specifications, and limitations. The work can be useful for Java developers who seek to choose best standards. Finally, the authors declare SUN and Google standard as best standards a PMD as the best tool.

E. Exceptions

Exception is an unintended complexity in a software system, in which the program handles particular errors that arise with exclusive exceptions. There are errors that occur by throwing or not throwing exceptions and many tools are used to detect such types of errors. JLint [20] it checks code for deadlocks, race conditions, redundant and suspicious calculations. Doctor J [11] Compare documentation against code like misspelled words, exception names parameter. Facebook Infer [10] checks for null pointer resource leaks, exceptions, annotation reachability, missing lock guards, and concurrency race conditions in code.

Some prominent work on dependency rules is done by X. Wu and J. Wei [1]. Developing healthy and secure software is always a challenge, so for this purpose programming languages like java and C++ introduces concept of exception handling. In this paper, authors emphasize that even after use of exception handling bugs are still remnant in the code. The paper discusses automatic violation detection and proposes new bug pattern and unsafe use of variable. In this paper, an algorithm has been described which is used to find out unsafe use of variables in programs. More work on dependency rules is done by F. Mariya and D. Barkhas [3]. For designing new software and also for checking quality of existing software tests mutation analysis technique is utilized. In this technique, some program configuration is done at small level and comparative analysis of two Java mutation testing tools has been presented. The two tools are: Pitest and μJava. Exceptions are introduced to check the performance of these two tools. From comparison, it is concluded that Pitest is easy to learn and is actively supported by developers. Pitest lists of libraries are regularly updated. Pitest is much faster in performing its functions.

F. Compatibility

Compatibility is a very complex issue. There are three types of possible incompatibilities concerning to a release of the Java platform that are: Source (compatibility related to translating Java source code into class files), Binary (compatibility is related to change to a type in binary that is compatible with pre-existing binaries) and Behavioral (compatibility related to the semantics of the code that is executed at runtime). Clirr [5] is a tool which checks binary compatibility of Java release with older releases.

G. General

It can be defined as an error, bugs in computer code that produces an invalid result. Bugs may have subtle effects or cause the program to crash system. In general bugs, incorrect or misspelled words, performance issues, general rules and naming conventions are included. FindBugs [13], Checkstyle [1], Coqua [7], DoctorJ [11], ckjm [3], SonarJava [23], Macker [21], Q-JPro [22], Spoon [24], coverity [8], and Facebook finder [10], PMD [2] are tools that check program against general bugs. FinBug [13] is most important tool for checking code against general errors. Checkstyle [1] checks white spaces, braces, naming conventions in program. With the help of these tools general errors are removed from code.

General rules checking tools are analyzed in a work [2].In this paper, a new tool JSC (Java Static Checker) has been described. The tool checks Java code for defective or flaw class based on propagating values and examining fault classes. The tool also checks general bugs in code. JSC is static tool and standard optimization algorithms are used by JSC. JSC is a lightweight tool. JSC checks semantics and syntax of program for possible defects without proving fault absence. The paper also carries out a comparison of FindBug, JLint and JSC. In [4], a case study has been discussed. Students used different open source tools like PMD, JLint, FindBugs and JSC. Comparison has been done and from comparison it is concluded that for improving maintenance of code PMD is best tool. JLint is helpful tool for finding issues that are related to inheritances. JSC accurately finds between four to eight times more nuIlpointer dereferences with false positive rate very low. FindBugs is useful tool to locate performance issues for nearly half of the reported faults. Widely used Static tools in current era are discussed in [6]. The authors have tried to classify many types of tools that are for different programming languages and different purposes according to

numerous categories like: availability of rules, extensibility, supported languages, technology and numerous other categories. The paper is aimed at building taxonomy of tools. At the end of paper, a comparison has been performed that is based on the established tree and on the presented static code analysis tools.

H. Security

Software security is an important software attribute, to prevent unauthorized person from using or accessing software or its data. Security bugs are errors that can be used to acquire unauthorized access on system. Security bugs introduce security vulnerabilities. Tools which check security are: DefenseCodeThunderScan [9], GrammaTech [11], HP Fortify Software [12], IBM Security AppScan [13], Kiuwan [14], Yasca [18], Lint4j [20], Checkmarx SAST Static Code Analyser [5], Cigital SecureAssist [6], Codacy [7] and Coverity [8]. Yasca [18] checks the code against security vulnerabilities. All tools described checks against security vulnerabilities.

Security tools are analyzed in [5]. In this paper, different tools are analyzed from security point of view and their results are presented in the graphical form that shows the weakness and strength of different tools. These results are obtained from a case study involving the participants who ran defined test cases on 14 different static analyzers. It was realized from the results that tools had considerably different support for most bug classes. The authors suggest that there should be focus of users on tradeoffs when they are selecting tools for their particular purpose because no tool succeeds in all respects. The authors conclude that code complexity directly affect the capability of tools to find defects. Bugs in simpler codes were easy to find than in complex codes. In [12], findings of research on source code analysis tools which was carried out at CERN is presented. 30 different tools were compared and analyzed presenting programmers with a list of efficient tools from which they can choose according to their requirements. Tools for C/C++, Java, Perl, Python and PHP languages were selected as per recommendation of Computer Society Team. The main focus or the comparison was based on tools that provide better security.

V. CONCLUSIONS

In static analysis, software program's source code can be analyzed manually or using automated techniques. Efficiency and quality in detecting software defects can be achieved using automated identification in the source code via diverse tools. These tools are developed by using many approaches such as data flow inspection and Taint Analysis. These tools are less or more successful in locating errors related to rules or standards in software code. In this paper, we look at what the Java code analyzers can do. We analyze many Java code analyzers or tools, some of these tools help to find and detect potential bugs and some of them check coding standards and metrics. An extensive analysis of the tools and literature for Java code analyzers has led to a consolidated review that has been presented in this study. This review would be highly beneficial for the researchers and scientific community targeting tool development for Java based source codes.

REFERENCES

[1] X. Wu and J. Wei, "Static Detection Unsafe Use of variables In Java," *7th International Conference on Ubiquitous Intelligence & Computing and 7th International Conference on Autonomic & Trusted Computing*, Xian, Shaanxi, 2010, pp. 439-443.

[2] S. Schmeelk, "Static checking Java with the Java Static Checker," *2nd International Conference on Software Technology and Engineering*, San Juan, PR, 2010, pp. V1-222-V1-226.

[3] F. Mariya and D. Barkhas, "A comparative analysis of mutation testing tools for Java," *IEEE East-West Design & Test Symposium (EWDTS)*, Yerevan, 2016, pp. 1-3.

[4] S. Schmeelk, "Open source programming assistance for students," *International Conference on Educational and Information Technology*, Chongqing, 2010, pp. V1-538-V1-545.

[5] Delaitre, B. Stivalet, E. Fong and V. Okun, *"Evaluating Bug Finders -- Test and Measurement of Static Code Analyzers,"* IEEE/ACM 1st International Workshop on Complex Faults and Failures in Large Software Systems (COUFLESS), *Florence, 2015, pp. 14-20.*

[6] J. Novak, A. Krajnc and R. Žontar, "Taxonomy of static code analysis tools," *The 33rd International Convention MIPRO*, Opatija, Croatia, 2010, pp. 418-422.

[7] N. Manzoor, H. Munir and M. Moayyed, "Comparison of Static Analysis Tools for Finding Concurrency Bugs,"*IEEE 23rd International Symposium on Software Reliability Engineering Workshops*, Dallas, TX, 2012, pp. 129-133.

[8] P. Tomas, M. J. Escalona, and M. Mejias, "Open source tools for measuring the Internal Quality of Java software products. A survey," *Computer Standards and Interfaces*, 2013, pp. 244–255.

[9] Z. Bougroun, A. Zeearaoui and T. Bouchentouf, "Comparative Study of the Quality Assessment Tools Based on a Model: Sonar, Squale, EvalMetrics," *Journal of Computer Science,* 2016, pp.39-47.

[10] Aftandilian, E., Sauciuc, R., Priya, S. and Krishnan, September. "Building Useful Program Analysis Tools Using an Extensible Java Compiler", *In Source Code Analysis and Manipulation (SCAM), IEEE 12th International Working Conference,* 2012, pp.14-33.

[11] M. A. Abdallah, and M. M. Al-Rifaee. "Java Standards: A Comparative Study," *International Journal of Computer Science and Software Engineering (IJCSSE*), 2017, pp146.

[12] T. Hofer (2017). [online] Available at: https://infoscience.epfl.ch/record/153107/files/ESSCAT-report-for-printing.pdf [Accessed 25 Sep. 2017].

[13] CH, A. (2017). Java Tools: Source Code Optimization and Analysis. *[online] Java Code Geeks. Available at: https://www.javacodegeeks.com/2011/07/java-tools-source-code-optimization-and.html [Accessed 25 Sep. 2017].*

[14] Anon, (2017). [online] Available at: https://infoscience.epfl.ch/record/153107/files/ESSCAT-report-for-printing.pdf [Accessed 25 Sep. 2017].

[15] En.wikipedia.org. (2017). *List of tools for static code analysis.* [online] Available at:https://en.wikipedia.org/wiki/List_of_tools_for_static_code_analysis [Accessed 25 Sep. 2017].

[16] Java-source.net. (2017). *Open Source Code Analyzers in Java.* [online] Available at: https://Java-source.net/open-source/code-analyzers [Accessed 25 Sep. 2017].

[17] Anon, (2017). [online] Available at: https://www.cs.montana.edu/~agata.gruza/papers/ESOF%20522.pdf [Accessed 25 Sep. 2017].