Tuijin Jishu/Journal of Propulsion Technology
ISSN: 1001-4055
Vol. 44 No. 4 (2023)

Evaluating CB measures-based Code Analyser

A.P.D.N De Vass Gunawardane[1], K.C.S.Madhumalka[2], Dr.D. I. De Silva[3], Tharushi Lakshika V.G[4], Samarasinghe.V.R[5], M. V. N. Godapitiya[6]

Faculty of Computing, Sri Lanka Institute of Information Technology, Malabe, Sri Lanka

Abstract: -Software complexity measures serve both as an analyser and a predictor in quantitative software engineering. Software quality is defined as completeness, correctness, consistency, no misinterpretation, and no ambiguity, feasible and verifiable in both specification and implementation. To better develop program optimisation a program, developers could use different methods, one being Complexity measures. This method is helpful for code debugging without compiling or running the program. It could also be automated using tools, which is considered more efficient than manually analysing the code. One specific method commonly used in automated code analysis is Cognitive-based complexity measures. There are many complexity metrics, and we concede the Cognitive-based measures introduced by Chhillar and Bhasin. This web-based automated code analysis tool based on Improved Cognitive Base measures unlikely other code analysis tools. This tool can detect duplicated code segments and, after seeing those code segments, calculate the complexity of those caught code segments. This will allow developers to increase software quality and increase optimisation.

Keywords*: Software Development Cycle, Improved Cognitive based, Automated Code Analyser, Static Code Source Code Analysis, Object Oriented.*

1. Introduction

The Secure Software Development Cycle (SSDC) extensively used Source Code Static Code Analysis (SCSA)[1] and Improved Cognitive Based (ICB) [2] Complexity measure. These methods are widely used for producing analysis reports, highlighting specific violations of the established code quality standards. In addition to producing a report, static code analysis tools and code complexity measures assist software engineers by outlining the root cause of a specific fault and how it might be fixed.

Larger teams and more lines of code are needed for the increasingly sophisticated modern software projects. Delivering durable, safe, and efficient software solutions is more critical than ever in today's fast-paced and changing technological world. Security and performance flaws, in contrast to compile-time defects, can make an application execute slowly in both the storage and compile aspects.

Like performance flaws, security bugs can be somewhat expensive to fix as the application's level of development improves. Even an experienced software developer would find performing source code manually futile since it takes a lot of time and effort, and there is still a chance that a security problem may exist.

These kinds of problems can be solved using ICB measures and SCSA ensures that the code follows consistent style rules by enforcing coding standards and best practices. Code becomes cleaner, more understandable, and easier to maintain even after software products are released. In general, finding and solving problems in the Software Process Cycle (SPC) is less expensive than waiting after compiling the codes. Automated code analysis (ACA)[3] and ICB measure matrices help to reduce the requirement for pricey maintenance and bug patches after release.

Duplicated code detection, SCSA, ACA, and ICB complexity measures may benefit from further optimisation, increased performance, and improved code quality[4]. These measures specifically help the Secure Software Development Cycle (SSDC).

In conclusion, these matrices, also known as static analysis tools, are crucial elements of contemporary software development. These tools work without code execution and are intended to identify duplicate code lines, vulnerabilities, and possible issues in the source code.

In the past history[2] of the complexity metrics which have been proposed mainly based on ICB measures. The methodology conducted in the next section proposed how to measure duplicated code and optimise the code.

2. Background

A. Software Metrics and Code Analysis

In the Secure Software Development Cycle (SSDC), software metrics, Source Code Static Code Analytics (SCSA), and Automated Code Analyzer help to understand the software project better, find bugs, optimise, and evaluate software projects and resources against established standards, and goals [5]. This software development and its growing emphasis on improving software quality assessment. This motivation is rooted in the acknowledgement, as expressed by DeMarco [2] and echoed in the industry, that the ability to measure and manage software holds significant importance.

For technical and management decision-making, Software Metrics, Automated Code Analyzers (ACA), and Source Code Static Code Analysis (SCSA) are helpful tools. These methods enable various measurements and various analysis aspects such as can be used for code optimisation, quality of final software product, complexity, reliability, testability, etc., and bug fixes and maintenance much more reliable, efficient, and cost-effective.

There are many definitions of Software Metrics, including the following,

- Software Metrics can be used in quantity a software as well as the process that is used to produce it [6].
- Software metrics can be used to quantify attributes which are derived from development processes, products and supporting resources [6].

ACA assesses Source Code based on four major metric categories, namely. [3]

- Object Oriented (OO) Metrics
- Size Metrics
- Complexity Metrics
- Maintainability Metrics

In Source Code, Static Code Analysis provides various advantages such as the following, [7]

- Find Bugs
- Performance Optimization
- More cost-effective

Software complexity measures like ICB measure, SCSA, and ACA serve both as an analyser and a predictor in quantitative software engineering. In now lots of CB measures are already exist Ex,

- Cyclomatic Complexity (CC)
- Halstead Metrics
- Class Complexity
- Weighted Class Complexity (WCC)[8]
- Object Oriented Metrics[3]
- Size Metrics[3]
- Maintainability Metrics [3]

Some of those metrics. all these metrics and Analysis tools could be used for program optimisation, Debugging, evaluation, and finding errors even before running the program in some scenarios.

B. Software Metrics and Code Analysis

Weyuker has suggested nine properties, which are used to determine the effectiveness of various software complexity measures[4]. A good Complexity measure should satisfy most of the Weyuker properties. A novel complexity measure that may assess complexity from both the architectural and cognitive perspectives has been created and presented in [9] and is based on the cognitive weights of the program. This study attempts to assess this metric as a good and thorough one by comparing it to the nine Weyuker properties [10].

This research initiated the study of ACA, SCSA and ICB measures, using these analysis methods to implement software analysis tools for software developers. While other code analysis tools do not provide duplicated code detection, Code optimiser and ICB Metrics.

We are proposing new metrics called Duplicated Code for ICB measure. Duplicated code lines increase cognitive load because of the need to find those duplicated code lines andremove or maintain duplicated code segments. In the Software Development Process, especially in the Development Process (Implementation Process), many mistakes like Syntax Errors, Code Segment Duplication, and many more developers cause these mistakes in the program to run with errors and bugs and less efficiently.

Using SCSA and ACA can detect duplicated code segments, and it is more efficient and reliable because if these are done manually, it takes time and more effort. Helps with AI tools and ICB measures possible to create more optimal software codes.

Weyuker has suggested nine properties, which are used to determine the effectiveness of various software complexity measures. A good Complexity measure should satisfy most of the Weyuker properties[4]. A novel complexity measure that may assess complexity from both the architectural and cognitive perspectives has been created and presented in [9] and is based on the cognitive weights of the program. This study attempts to assess this metric as a good and thorough one by comparing it to the nine Weyuker properties [10].

C. Software Static Code Analytics Method

3. ICB Measure

A. ICB Based Application

This analysis web application should support every JAVA-based program code, including JAVA OOP program codes. Helps of ICB metrics, ACA and SCSA implemented Code Analysis web application capable to,

- Measure the complexity of the code.
- Find syntax errors.
- Find duplicated code segments.
- Optimist the code.

The CB measure focuses on four factors as follows,[8],[9],[2]

- Nesting level of control structure (Wn).
- Type of control structure in class or program (Wc).
- Inheritance level of statement in classes (Wi).
- Size of a class or program regarding token count.

Helps these four factors calculate the complexity of an object-oriented program.

$$C_w(P) = \sum_{j-1}^{n} (S_j) \times (W_t)_j \qquad (1)$$

C_w (P) = Proposed Weighted Complexity measure of the program

S_j = Size of j^{th} executable statement in terms of token count

n = Total number of executable statements in the program

j = Index variable

$W_t = W_n + W_c + W_i$

Using CB values can measure the complexity of statements of the programs and can determine how much-duplicated code segments affect the overall program.

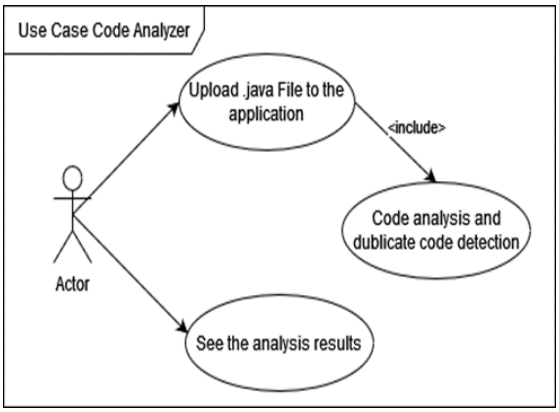
SCSA helps to find syntax errors and inspect code details (ex, Total number of classes, number of variables, for loops, if-else condition, etc.).

In duplicated code segments (DCS), detection could optimise the program and remove unnecessary code segments. It uses an algorithm to detect duplicate codes inside the program. After detecting those duplicated code segments, we could measure the complexity of those specific code segments, which helps to determine how much affected the duplicated code segments.

Code optimisation is crucial for better performance and increased readability. Helps with the syntax tree and removing DCS. Even code optimisation helps to improve user experiences and responsiveness.

B. Functional Requirements

The code analysis tool build has 4 main functionalities, as drawn in the use case diagram in Figure 2. First, the user could upload their JAVA code project into the tool as a .java file format. The tool will analyse the project and then show the results to the user. The result set includes ICB measures, if there are syntax errors, those errors will show up, as duplicated code segments—Optimise code suggestions.



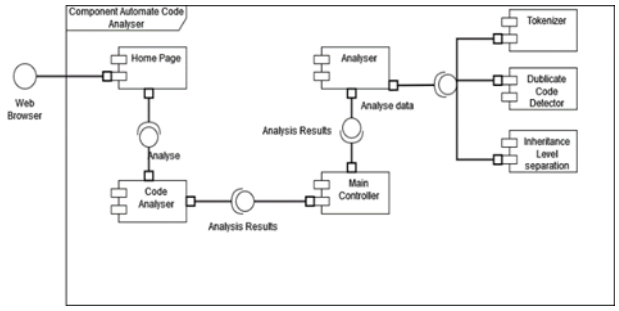
C. Workflow and Architecture

The equations are an exception to the prescribed specifications of this template. To calculate ICB metrics and detect duplicated code lines in the JAVA code/ program, the tool would parse the JAVA code to analyse routes after getting the project from the user. Then, the JAVA code breaks down into several levels: inside those levels, the code breaks into tokens. Using each token count, calculate the complexity of the regulation (ICB Measures).

After calculating the ICB measure, the analysis tool will check the duplicated code segments in the JAVA code. Its analysis happens to help code patterns and the algorithm. The tool will check that each line of code segment matches another line of code segment. If both code segments match, it will highlight as a duplicated code segment. Then again, ICB metrics will be used to analyse the complexity of the duplicated code segment.

Lastly, the analysis tool will check the syntax errors, and generate a summary of the JAVA code help of Static Code Source Code Analysis methods. This report will help programs to understand the code better and for the optimisation process.

The analysis tool has six central components: Tokenizer, Inheritance Level separation, ICB analyser, Duplicate code detector, and code analyser. The component diagram is shown in Figure 3. The tool is implemented using



React and JavaScript languages and the MERN stack web framework.

a) Home Page:

The Home page is a component which provides a browser view to the users, where they could upload their JAVA code files for analysis. This component passes the upload code to Code Analyser and receives the analysis results from Code Analyser. Then, it is displayed.

b) Code Analyser:

This component is the main part of the application which also connects the Home Page and the main controller.

c) Main Controller:

The Main Controller is the main program in the application it passes the code to the analyser to analyse the code, and the central controller receives all analysis data from the analyser which is pass to Code Analyser.

d) Analyser:

The analyser component does the backend process of the application. After receiving the code from the central controller component, it calculates the ICB measure of the code, duplicated code segments, count number classes, and if-else statements for loops like vice. These calculations are done with the help of other 3 components called Tokenizer, Duplicated code detector and Inheritance level separator. ICB metrics are calculated using the Fig. 1 formula.

e) Tokenizer:

The tokenizer component involves separating every keyword inside the java program.

f) Duplicated Code Detector:

The duplicated code detector component is involved in finding duplicated code segments inside the Java program. If any duplicated code is found, those results will be sent back to the analyser component.

g) Inheritance level separation:

This component is very important for calculating complexity of the java program. its can separate each level of the program.

4. Improvements and Evaluation

The main improvements of this analysis tool are finding duplicated code segments, calculating the complexity of those code segments, and optimizing the code. In the past, only analysis, was the complexity of complete program code, but in this code analysis tool is capable of calculating complexity of full code and if their duplicated code segments that code segment's complexity. It can find syntax errors and code detail summarizing.

5. Conclusion

We have discussed the vital role of software complexity measures, with a specific focus on Improved Cognitive Based (ICB) measures, in the realm of quantitative software engineering. The importance of software quality, encompassing attributes such as correctness, consistency, and maintainability, has been emphasized. As software projects grow in complexity and importance, the need for effective tools and methodologies to ensure code quality and security has become increasingly evident.

The challenges faced in modern software development, including compile-time defects, security vulnerabilities, and performance bottlenecks, necessitate proactive and efficient solutions. Traditional manual code analysis methods are not only time-consuming but also prone to human error. Therefore, the adoption of automated code analysis tools is imperative to streamline the development process and enhance software quality.

Refrences

[1] P. Mezhuev, A. Gerasimov, P. Privalov, and V. Butkevich, "A dynamic algorithm for source code static analysis," Proc. - 2021 IvannikovMeml. Work. IVMEM 2021, pp. 57–60, 2021, doi: 10.1109/IVMEM53963.2021.00016.

[2] D. I. De Silva, N. Kodagoda, S. R. Kodituwakku, and A. J. Pinidiyaarachchi, "Improvements to a complexity metric: CB measure," 2015 IEEE 10th Int. Conf. Ind. Inf. Syst. ICIIS 2015 - Conf. Proc., pp. 401–406, 2016, doi: 10.1109/ICIINFS.2015.7399045.

[3] H. M. Fernando, D. R. Kothalawala, D. I. De Silva, and N. Kodagoda, "Automated code analyser," Proc. IASTED Int. Conf. Eng. Appl. Sci. EAS 2012, pp. 196–201, 2012, doi: 10.2316/P.2012.785-109.

[4] S. Misra and A. K. Misra, "Evaluating cognitive complexity measure with weyuker properties," Proc. Third IEEE Int. Conf. Cogn. Informatics, ICCI 2004, no. 3, pp. 103–108, 2004, doi: 10.1109/COGINF.2004.1327464.

[5] J. Yang, C. Barrientes, J. Sanchez, and Y. R. Kim, "Source code analysis for secure programming practices," Proc. - 2018 Int. Conf. Comput. Sci. Comput. Intell. CSCI 2018, pp. 819–824, 2018, doi: 10.1109/CSCI46756.2018.00164

[6] Y. Wang, "Cognitive complexity of software and its measurement," Proc. 5th IEEE Int. Conf. Cogn. Informatics, ICCI 2006, vol. 1, pp. 226–235, 2006, doi: 10.1109/COGINF.2006.365701

[7] D. Binkley, "Source code analysis: A road map," FoSE 2007 Futur. Softw. Eng., pp. 104–119, 2007, doi: 10.1109/FOSE.2007.27.

[8] D. I. De Silva, N. Kodagoda, S. R. Kodituwakku, and A. J. Pinidiyaarachchi, "Limitations of an object-oriented metric: Weighted complexity measure," Proc. IEEE Int. Conf. Softw. Eng. Serv. Sci. ICSESS, vol. 2015-Novem, pp. 698–701, 2015, doi: 10.1109/ICSESS.2015.7339153

[9] D. I. De Silva, "Enhancements to an OO Metric: CB Measure," J. Softw., vol. 12, no. 12, pp. 72–81, 2018, doi: 10.17706/jsw.13.1.72-81

[10] eyuker, E., Evaluating software complexity measure. IEEE Transaction on Software Complexity Measure, 14(9):1357–1365, September 1988