RESEARCH ARTICLE OPEN ACCESS

SICCAT:Software Inheritance Coupling Complexity Analysis Tool

Vanitha N[1], ThirumalaiSelvi R[2]

[1](Research Scholar, Research and Development Centre, Bharathiar University, Coimbatore, India

& Assistant Professor,Department of Computer Science, Women's Christian College, Chennai, India)

[2](Assistant Professor, Department of Computer Science, Govt. Arts College for Men, Nandanam, Chennai, India)

Abstract:

Software developers engage themselves to develop better software in time. Software quality is achieved when the software is free of defect, timely delivers within a specified budget, and meets customer requirements and ease to maintain. One of the approaches to reduce the software defect is to identify the part of the program that render high coupling which effects software quality. The changes can be done in the portion of the program where the error is likely to occur due to its dependency among the modules and thus put forth some effort on the overall quality of the software artifact. The aim of this paper is to design and develop a software tool that finds the software complexity that may take place due to inheritance coupling at the package level and visualize the same in graph. Based upon the complexity value the developers can change the code at the earlier phase itself and thus lower the complexity to some extent. This will help the software developers to minimize their time and cost for the development of quality software.

***Keywords* — Complexity, Metrics, Tool, Visualization.**

I. INTRODUCTION

Software complexity stands for the measure of an entity, which have extremely complicated structure and so many interconnected links. During the design phase, the number of interconnections between the elements increases gradually, which turns into very hard to understand and as the software complexity affects many quality attributes such as reliability, usability, modifiability, etc. either directly or indirectly, it is the responsibility of software developers to build low complex software products. So without any complexity measure it is difficult for the developers to assess the project complexity. In Java based projects, Coupling is one of the important metrics in measuring the software complexity that affects the overall software performance. Tight coupling subsists between two classes when one class is collaborating with another class .Inheritance Coupling occurs when one class inherits the property of another class within and between Software quality. Using coupling metrics one can examine and compare the data with the previous result and ensure the improvements happened. A software tool is a program that helps to develop applications in a proper and easy manner. So many software defect estimation tools have been effectively applied in many real-world software projects. The purpose of this paper is to develop a software tools that calculates the Inheritance coupling complexity of the product on a single click which aims to reduce the time and effort of the developers.

This paper measures the complexity of the object oriented software system using inheritance coupling at the package level, which uses the CK metrics [1] such as Coupling between Objects (CBO), Depth of Inheritance Tree (DIT) and Number of Children (NOC) to identify and analyze the coupled components. The complexity is measured by assigning different weights based on the dependencies between components at the package level. The main goal of this paper is to design and develop a software tool that computes the software complexity that occurs due to inheritance coupling at package level. Once computes the complexity, identifies the most error prone classes and make changes a in such classes, thus reduce the complexity. This study uses CK metrics to compute the complexity. As visualization is a useful and easy way of depicting the results to the user, this paper aids the developers to visualize their results.

The rest of this paper is structured as follows. *Section II* presents literature review *Section III discusses* the background to the field. *Section IV* provides the architecture of the newly proposed tool. *Section V* gives detailed description of the newly proposed tool. *Section VI* presents the study results. *Section VII* concludes the paper and summarizes some future work.

II. LITERATURE REVIEW

A study of Conference proceedings or Journal paper published recently or only the most comprehensive to be included in this review. Included papers were published between the year 2005 and 2018. Google Scholar, Scopus, IEEExplore and ScienceDirect were the search engines covered most of the publications given in references. Totally 42 papers were identified, 19 unrelated papers to this research were rejected and finally included 23 papers.

Reference [2] presented a software visual analytics tools which analyze static program and extract the fact and visualize the information which support program comprehension. Reference [3] shows the tool developed for analysing and visualizing the networks. [4] merged the code complexity with the maintenance time.

III. BRIEF DESCRIPTION ABOUT THE PREVIOUS STUDY

A. *ICCP Metric*

Software complexity can be estimated using various factors of software. Coupling is one of the factors which affect the quality of software. In this section, we are discussing a newly proposed metric and a tool for measuring the complexity of the software system based on the inheritance coupling at the package level and visualizes the same in graph.

As shown in [1] deeper the level, higher the value of the DIT. Deeper trees involve greater design complexity with a higher probability of errors in the code.

According to the previous study [5] inheritance of classes from sub package in one package to sub package in another package is at a deeper level of inheritance which is harder to understand and thus gives more complexity than else. Based on the inheritance at package level weights are assigned. Deeper the inheritance level higher the weights. The assignment of different weights based on the inheritance of the classes at the package level from 1 to 5 is shown in the following Table 1. No weight has assigned to the state in which the packages could not share any classes and it is mentioned as not applicable (NA) in Table 1.

TABLE I
ASSIGNMENT OF WEIGHTS

Inherit Classes	Same Package	Different Package	Same Sub Package	Different Sub package
Same Package	1	NA	2	4
Different Package	NA	1	4	2
Same Sub package	2	4	3	5
Different Sub package	4	2	5	3

The data collected from different Java projects have chosen to demonstrate the practical features of the proposed tool. The total number of parent packages, its sub packages and classes in the five projects are shown in the Table 2.

TABLE III
NUMBER OF PACKAGES AND CLASSSES IN FIVE DIFFERENT JAVA PROJECTS

Java Projects	**Library Management System (Project 1)**	**External Remuneration System (Project 2)**	**Stock Maintenance System (Project 3)**	**Student Result System (Project 4)**	**Online Voting System (Project 5)**
No. of Parent packages	4	2	2	3	2

No. of Sub Packages	10	6	4	9	5
No. of Classes	42	31	26	53	20

The number of classes inherited at package level in a different manner for the five projects are shown in the Table 3.

TABLE IIIII
INHERITANCE COUPLED CLASSES AT PACKAGE LEVEL FOR FIVE PROJECTS

Coupled Classes at Package level	**Project 1**	**Project 2**	**Project 3**	**Project 4**	**Project 5**
Classes extends from the same package	5	2	3	10	4
Classes in sub package extends from the same package	3	2	7	12	4
Classes in sub package extends classes of sub package belongs to same package	6	4	3	5	2
Classes in package extends classes of sub package belongs to other package	4	7	2	6	0
Classes in sub package extends classes of sub package belongs to other package	4	7	0	5	5

We have chosen the mid-range formula Eq. (1), for calculating the lowest and highest possibility of Inheritance Coupling complexity values.

Mid-Range formula is a set of statistical data values is the arithmetic mean of the maximum and minimum values in a data set and it is a measure of central tendency.

$$\textit{mid-range} = \min + (\max - \min) / 2 \qquad (1)$$

mid-range – expected complexity value
min – minimum complexity value that can be occurred
max – maximum complexity value that can be occurred

From Table 4 it is shown that the difference between the actual complexity value and expected complexity value are high for the projects 2, 3 and 4 and hence the probability of getting risk is high which may diminish overall software quality.

From the results, it has shown that the number of the class inherits in the deeper level is more results high complexity.

Projects	Actual Value	Expected Value
Library Management System	65	66
External Remuneration System	81	66
Stock Maintenance System	34	45
Student Result System	98	114
Online Voting System	43	45

TABLE IV
ACTUAL AND EXPECTED VALUE OF INHERITANCE COUPLING COMPLEXITY

B. Proposed Tool

From the previous study, we proposed a tool which is named as Software Inheritance Coupling Complexity Analysis tool (SICCAT) which can be divided into complexity Evaluation and visualization, as follows. Complexity Evaluation collects the information about the source code from the database and computes the complexity in the proposed manner. Visualization uses information from the database and use visualization techniques to create interactive displays of the software complexities in graph. SICCAT uses the ICCP algorithm [5] for the complexity evaluation. Fig. 1. Shows the architecture of newly proposed SICCAT tool

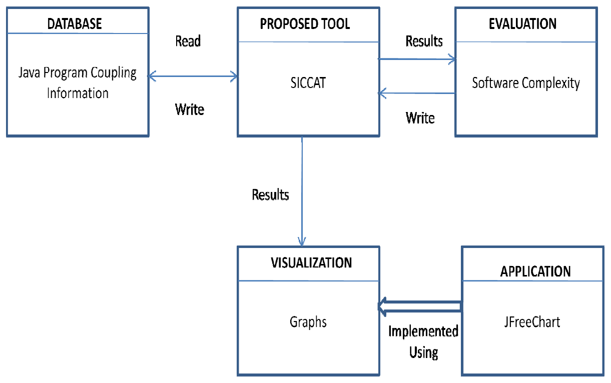
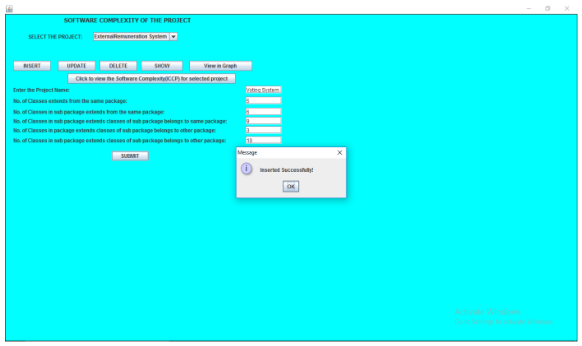


Fig. 1 Architecture of newly proposed SICCAT tool

We have built the Software Inheritance Coupling Complexity Analysis Tool (SICCAT) that calculates the complexity of software product that may arise due to inheritance coupling at package level that was developed in Java and collaborates with other software applications, including a JFreeChart and WAMP Server. SICCAT accepts the input from the user needed to compute the complexity and stores it in the database. SICCAT retrieves the information whenever needed from the database and allow developers to update and delete the data if they want. The tool not only computes the actual complexity, but also assists the developers in determining whether to reduce the complexity level or not, by making changes in the defect prone code by comparing the actual and desired complexity value in tabular form and also visually. The actual and desired complexity values are calculated according to the previous work [5]. The calculated complexity values are layout into tabular forms and saved back into the database.

Fig. 2. Shows SICCAT's screen design, which has the option to insert, update, delete and view the data's into and from database.


SOFTWARE COMPLEXITY OF THE PROJECT
INSERT
UPDATE
DELETE
SHOW
View in Graph
Click to view the Software Complexity(ICCP) for selected project
SUBMIT
OK

SELECT THE PROJECT: ExternalRemuneration System

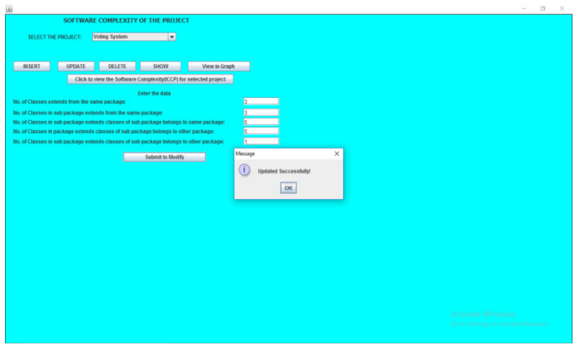
Enter the Project Name:	Voting System
No. of Classes extends from the same package:	5
No. of Classes in sub package extends from the same package:	6
No. of Classes in sub package extends classes of sub package belongs to same package:	9
No. of Classes in package extends classes of sub package belongs to other package:	3
No. of Classes in sub package extends classes of sub package belongs to other package:	10

Message

Inserted Successfully!

Fig. 2 Data inserted successfully into the database

Fig. 3. shows the software project, which is to be updated by choosing from the combobox by the user.


SOFTWARE COMPLEXITY OF THE PROJECT
INSERT
UPDATE
DELETE
SHOW
View in Graph
Click to view the Software Complexity(ICCP) for selected project
Submit to Modify
OK

SELECT THE PROJECT: Voting System

Enter the data

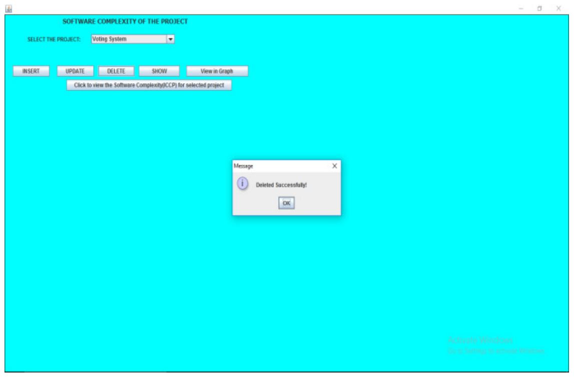
No. of Classes extends from the same package:	3
No. of Classes in sub package extends from the same package:	3
No. of Classes in sub package extends classes of sub package belongs to same package:	6
No. of Classes in package extends classes of sub package belongs to other package:	5
No. of Classes in sub package extends classes of sub package belongs to other package:	1

Message

Updated Successfully!

Fig. 3 Data updated successfully in the database

Fig. 4. Shows the software project deleted from the database successfully. The user has to select the project to be deleted from the combobox and when hits a delete button the project must be completely removed from the database.


SOFTWARE COMPLEXITY OF THE PROJECT
INSERT
UPDATE
DELETE
SHOW
View in Graph
Click to view the Software Complexity(ICCP) for selected project
OK

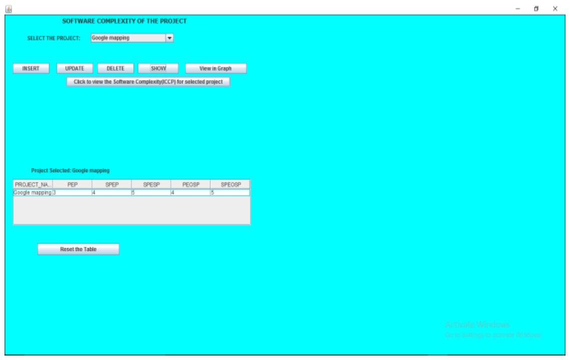
SELECT THE PROJECT: Voting System

Message

Deleted Successfully!

Fig. 4 Record deleted successfully from the database

Fig. 5. gives the information about the software project stored in the database when hits a show button.


SOFTWARE COMPLEXITY OF THE PROJECT
INSERT
UPDATE
DELETE
SHOW
View in Graph
Click to view the Software Complexity(ICCP) for selected project
Reset the Table

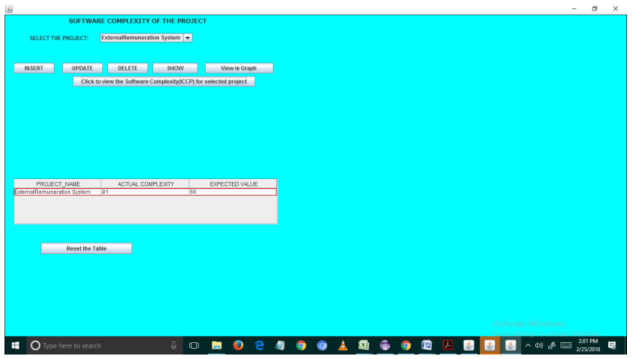
SELECT THE PROJECT: Google mapping

Project Selected: Google mapping

PROJECT_NA...	PEP	SPEP	SPESP	PEOSP	SPEOSP
Google mapping	3	4	5	4	5

Fig. 5 View details of software project in tabular format

Fig. 6. shows the actual and desired complexity value of the selected software project from the combobox in tabular format.


SOFTWARE COMPLEXITY OF THE PROJECT
INSERT
UPDATE
DELETE
SHOW
View in Graph
Click to view the Software Complexity(ICCP) for selected project
Reset the Table
Type here to search

SELECT THE PROJECT: ExternalRemuneration System

PROJECT_NAME	ACTUAL COMPLEXITY	EXPECTED VALUE
ExternalRemuneration System	81	66

Fig. 6 View actual and desired complexity values in tabular format

The view in Graph button tells SICCAT to show actual and desired complexity value in the graph. From Fig.7. it can be easily studied the acceptable level of complexity of the five projects.

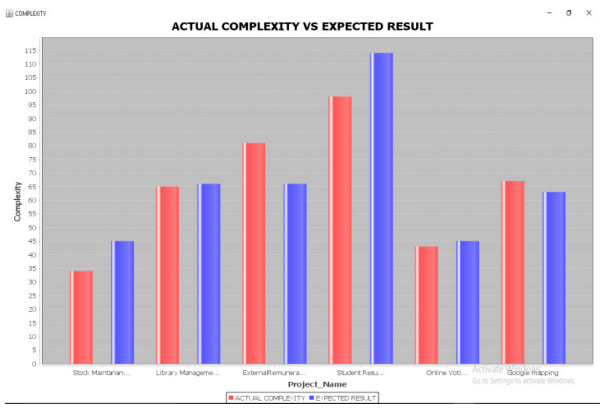


Fig. 7 Actual and expected values of Inheritance Coupling Complexity of different projects

IV. CONCLUSION AND FUTURE WORK

In this paper, we found the complexity of the object-oriented system with our newly proposed SICCAT tool. We have also associated our newly proposed inheritance coupling complexity metric with the software quality attribute, modifiability. The results have shown that more the numbers of the classes inherit in the deeper level results high complexity. Overall the results conclude that determining the complexity of the software program that may arise due to inheritance coupling using automated tool will help software developers and researchers to reduce their effort in terms of cost and time and to develop high-quality software. This study has used visualization techniques to easily discover the complex level of the software system.

In future, progress or extension of this study can be done by focusing on extending the tool for getting the software programs as the input, parse and obtain the information about the coupled classes and packages.

REFERENCES

1. S. Chidamber, and C. Kemerer, "Towards a metrics suite for object oriented design," ACM., vol. 26, no. 11, pp. 197–211 1991.
2. Reniers Voinea, L., Ersoy, O., & Tele a, A., "The Solid* toolset for software visual analytics of program structure and metrics comprehension: From research prototype to product," Science of Computer Programming., vol. 79, pp. 224-240.
3. N.J. Van Eck, and L. Waltman, "CitNetExplorer: A new software tool for analyzing and visualizing citation networks," Journal of Informetric.,, vol. 8, no. 4, pp. 802-823, 2014.
4. V. Antinyan, M. Staron, and A. Sandberg , "Evaluating code complexity triggers, use of complexity measures and the influence of code complexity on maintenance time,"Empirical Software Engineering., vol. 22, no.6, pp. 3057-3087, 2017.
5. N. Vanitha, and R.Thirumalaiselvi, "Inheritance Coupling Complexity Metric in Association with Modifiability at Package Level: An Empirical Exploration," International Journal of Pure and Applied Mathematics., vol. 118, no. 18, pp. 3789-3797, 2018.
6. J. Meinicke, T. Thüm, R. Schroter, F. Benduhn, and G. Saake, "An overview on analysis tools for software product lines,"In Proceedings of the 18th International Software Product Line Conference: Companion Volume for Workshops, Demonstrations and Tools-Vol. 2 , pp. 94-101, ACM, Sep. 2014.
7. P.W. McBurney, and C. McMillan, " Automatic source code summarization of context for java methods," IEEE Transactions on Software Engineering.,vol. 42, no. 2, pp. 103-119,2016.
8. T. Bakota, P. Hegedus, I. Siket, G. Ladanyi, and R. Ferenc, "Qualitygate SourceAudit: A tool for assessing the technical quality of software," In Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week-IEEE Conference on pp. 440-445,IEEE, Feb. 2014.

9. A. Poliakov, J. Foong, M. Brudno, and I. Dubchak, “GenomeVISTA—an integrated software package for whole-genome alignment and visualization,” Bioinformatics., vol. 30, no. 18, pp. 2654-2655,2014.

10. Ahrendt, W., Beckert, B., Bruns, D., Bubel, R., Gladisch, C., Grebing, S., ... & Mostowski, W, “The KeY platform for verification and analysis of Java programs,” In Working Conference on Verified Software: Theories, Tools, and Experiments , pp. 55-71 , Springer, Cham, Jul. 2014.

11. T. Mens, “Research trends in structural software complexity,”arXiv preprint arXiv:1608.01533, 2016.

12. V.S. Bidve, and P. Sarasu, “Coupling Measures and its Impact on Object-Oriented Software Quality” Indian Journal of Science and Technology., vol. 9, no. 21, 2016.

13. S. McIntosh, Y. Kamei, B. Adams, and A. E. Hassan ,“ An empirical study of the impact of modern code review practices on software quality,” Empirical Software Engineering., vol. 21, no. 5, pp. 2146-2189, 2016.

14. M. Burch, M. Raschke, A. Zeyfang, and D. Weiskopf, “A Scalable Visualization for Dynamic Data in Software System Hierarchies. In Software Visualization (VISSOFT),” IEEE Working Conference on pp. 85-93, Sep 2017.

15. S. Perer, “ Balancing systematic and flexible exploration of social networks”Visualization and Computer Graphics., IEEE Transactions on vol. 12, no. 5, pp. 693–700, 2006.

16. C.Mallikarjuna, K. Sudheer Babu, P. Chitti Babu, “A Report on the Analysis of Software Maintenance and Impact on Quality Factors,”International Journal of Engineering Sciences Research-IJESR. , vol. 05, Article 01335, 2014.

17. B.A. Price, R.M. Baecker, and I.S. Small, “ A principled taxonomy of software visualization” Journal of Visual Languages & Computing., vol. 4, no. 3, pp. 211-266, 1993.

18. F. Fittkau, A. Krause, and W. Hasselbring, “Software landscape and application visualization for system comprehension with ExplorViz” Information and software technology., Vol. 87, pp. 259-277, 2017.

19. N. Elmqvist, and J.S. Yi, “Patterns for visualization evaluation” Information Visualization., vol. 14, no. 3, pp. 250-269, 2015.