TECHNIUM

Technium Vol. 4, Issue 6 pp.84-98 (2022)
ISSN: 2668-778X
www.techniumscience.com

Constructing a Software Tool to Optimize Performance by Coupling Detection

Tawfeeq Mokdad Tawfeeq
Department of Software, College of Computer Sciences and Mathematics, University of Mosul, Mosul / Iraq
E-mail:tawfeeq.m.flaih@uomosul.edu.iq, ORCID: 0000-0002-2106-3604

ABSTRACT. Coupling relations reflect the interdependencies of software modules and can be used to assess a program's quality— Lower the coupling value will be, the higher the quality of the software will be. Coupling measures are crucial for determining the quality of object-oriented software, from design up to maintenance. Inside software engineering, quality attributes are realized non-functional requirements utilized to assess whether the software is of good quality or not, One of the quality attributes is Performance, which means the system can respond to various actions in a given time or how fast does it respond or execute. To calculate the total amount of time a program will require to run until completion use Time Complexity. In this paper a Computer Aided Software Engineering Tool has been constructed which is called CDPI (Coupling Detection to Performance Improvement). It parse a software source code to extract information about its structure, components, and relationships that connect its parts. That leads to improving the software readability, understandability and for detects coupling among classes of Java software.The CDPI tool detects coupling in software source code and Show it in table to the software engineer, To start the process of decoupling by the software engineer for the candidate coupling. The CDPI tool was tested by inputting software written in OOP by Java language. The CDPI Tool were evaluated by calculating the Time Complexity before and after decoupling. Results, source code after the decoupling is executed faster than before decoupling to produce the required output, which is an indication of Optimize Runtime Performance of the program.

Keywords. CASE Tool, Coupling, Software quality attributes, Runtime performance, Time Complexity.

1. Introduction

Software modularization is almost as old as the concept of software engineering itself [15]. Modularization is the method of splitting a source code into numerous independent modules, each of which operates independently [3]. Software engineering benefits from modularization in numerous ways. Among these is software that is simple to understand, software that is easy to maintain and a module that can be used multiple times without requiring rewriting [7][18]. Modularity is important, and Object-Oriented Programming is one of the ways of achieving it. OOP is a methodology or paradigm to design a

program using classes (which embed data and operations) and many relationships over these classes [13]. Object Oriented Analysis and Design is a well-known technique for designing and implementing software. OOP needs a distinct approach to software metrics in addition to a different approach to design and implementation[6][19]. These metrics focus on the architectural design of a software and to what degree does the modularization goal is achieved and can be used to assess the quality of an OOP, one of the terms used in OOP is coupling [5][17].

The notion of coupling is a measure of the degree of relatedness between modules of object-oriented software. Coupling measures play a significant role in the quality software [2]. Quality attributes are achieved by non-functional requirements. Higher the quality of the software will be, the Lower the coupling value. That's mean good software should have weak coupling between components and avoid tight coupling as much as possible [21].

High-quality software can be determined in part by its performance [24]. It is shows the responsiveness of a system to performe a required action with a set time frame [20]. To calculate the total time required by the program to run till its completion use Time Complexity method[16]. It is measures the number of time taken to execute each statement of code in an algorithm to produce the required output Independently from both hardware and software environment [23]. If a statement is set to execute repeatedly, the number of times it is executed is equal to N multiplied by the amount of time required to execute that function once [8].

Increased execution time, storage size, compilation and loading times, and inter-module communication issues are just a few of the drawbacks of unnecessary coupling [9]. That's mean decreasing the coupling in a software leads to an improved software performance [14][25]. For this reason we propose a CASE tool which is called CDPI (Coupling Detection to Performance Improvement) aims to detects coupling in OOP JAVA code. The CDPI tool candidate the detected coupling in a table and presents it to the software engineer, who in turn selects unnecessary coupling, and the tool will automatically decoupling it, and an Editing interface is displayed that contains the source code before and after decoupling. By calculating the value of Time complexity, it was found that the program's runtime performance has been improved.

This paper is organized as follow; the following section presents the related work, The proposed tool called CDPI is described in section 3, Section 4 covers testing the proposed tool and evaluating through case studies. The conclusions and possible future works are presented in section 5.

2. Literature Review

in this section we are going display a brief overview of a few existing approaches from the studies related to coupling measurement.

In 2010 [10], Gethers and Poshyvanyk proposed a new metric for object-oriented software called Relational Topic based Coupling (RTC), which uses Relational Topic Models (RTM) to capture latent topics in source code classes and their relationships. The new measure is compared to structural and conceptual coupling metrics using 13 open-source software systems. The results show that the authors' proposed metric expands on coupling dimensions and can effectively support software impression analysis. While in 2017 [4], Anwer el at study the impact of various types of coupling metrics on fault estimation by means of multivariate logistic linear regression. They examine the connection between these metrics and faults. They proved their model using many open source systems and the results show that the efferent coupling (Ce) is a well pointer for fault estimation than the other metrics like coupling between object and afferent coupling. In 2017 [1], Ajienka and Capiluppi analysis many open-source software to examine the relationship among the two types of coupling via many steps : , they measured the joining of class dependences, statistically calculated the association between class dependences, propose a method to fix the stability of open-source software, by gathering classes as "stable" or

Technium Vol. 4, Issue 6 pp.84-98 (2022)
ISSN: 2668-778X
www.techniumscience.com

"unstable", depends on their co-change design. The results show there is significant indication to determine that class pairs are coupled structurally contain logical dependences although there is no robust indication of a linear relationship among the powers of the coupling kinds. While in 2019 [22], Sousa el at study how OO software inner quality evolves in terms of coupling. The coupling conduct affects the complexity and reusability of the systems, and the ratio of classes from the systems directly affects the coupling development. The results show many features of coupling development, including the coupling conduct, is well modelled by a cubic task, coupling development increases software difficulty, systems tend to have a complicated design, and a small collection of classes exaggerates coupling development.

Lu el proposed MinClass in 2021[12], to study a few key Java classes. Initial class-level system construction is represented by a network reflecting coupling strength and road between classes. Next, they propose the OSE metric to rank software classes. The classes are sorted by OSE value, and a few are chosen as critical classes. According to research on Java projects, MinClass is the most valuable and promising software method .

3. Proposed Work

The previous studiesin section 2 were based on coupling detection in software to give an indicator to software engineers on coupling values to make early improvements without trying to do decoupling. So a CASE Tool has been proposed which is called CDPI (Coupling Detection to Performance Improvement) for extract an information in tables to improving the software readability and understandability and for detect coupling among classes of Java software, And presented it to the software engineer to choose the candidate couple by him to make a decoupling automatically.

The principle of the CDPI tool is to use (javaparser 1.0.8 jar) to interprets and analyzes the source code of the software to extract specific information that is essential in detect coupling among classes of object oriented Java softwar and displays them with a special table, The software engineer begins the process of decoupling with a accurate selection of the desired coupling to make decoupling after reading all information presented by the tool in the tables and making sure that it is a coupling must be decoupled, and the tool performs the decoupling automatically, Thats lead to reduce the impact of coupling on quality attributes then Optimize Runtime Performance. Figure (1) shows the block diagram of the CDPI tool.

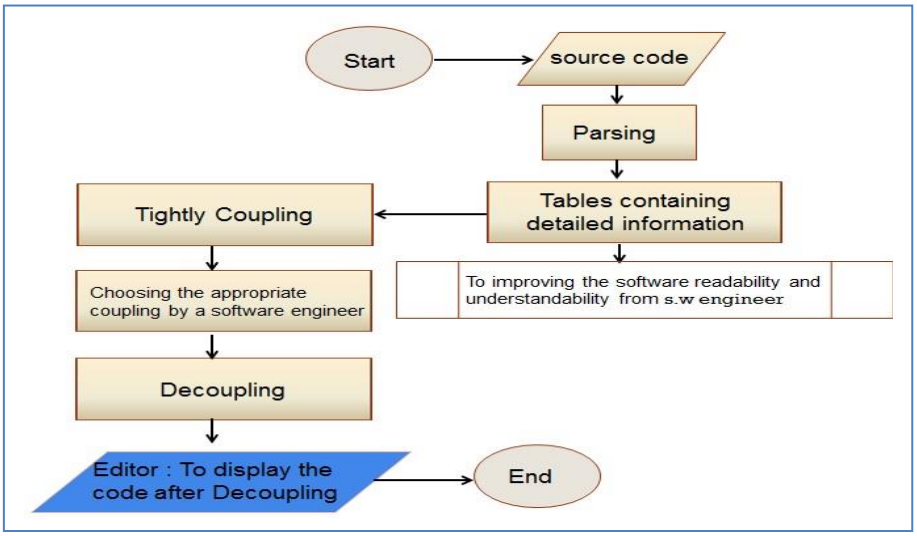

Figure 1. CDPI Tool Block Diagram.

Technium Vol. 4, Issue 6 pp.84-98 (2022)
ISSN: 2668-778X
www.techniumscience.com

When the CDPI tool is executed, six tables and a result display interface are produced, as follows:

1- Classes Tree: To know the packages and the component of each package and the types.
2- Operations: To know information about all operations of classes and interfaces.
3- Data : To give detailed information for all the variables in the classes and interface.
4- Objects: To extract the dependency relationship between classes or interfaces.
5- Inheratance: Determine the relationships between classes and interfaces.
6- Tightly Coupling: To help the software engineer choose the appropriate coupling to make decouple.
7- Decoupling Editing: It includes a right text box containing the source code chosen by the software engineer, and a left text box containing the decoupling of the chosen source code.

The UML Use Case Diagram (Level One) is used to describe the CDPI tool analysis phase, as shown in Figure (2).

.

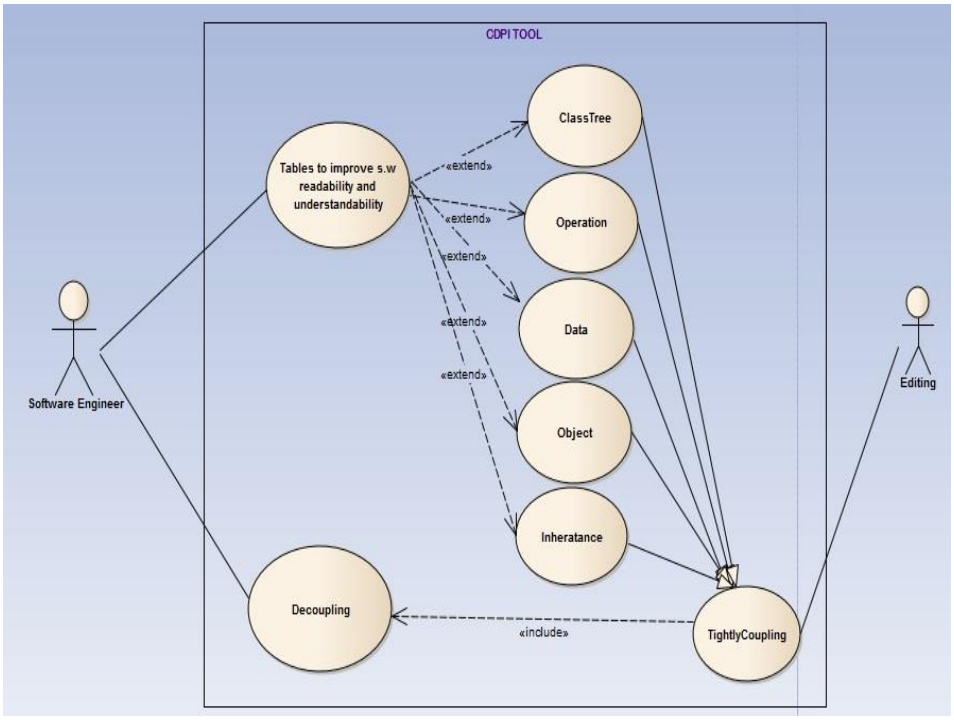

Figure 2. Use Case to CDPI Tool.

The UML Class diagram, which specifies the data and operations for each class, is used to show the design phase of the CDPI tool , as seen in Figure (3).

Then the time sequence of classes is determined using the UML Sequence diagram, as shown in Figure (4).

Technium Vol. 4, Issue 6 pp.84-98 (2022)
ISSN: 2668-778X
www.techniumscience.com

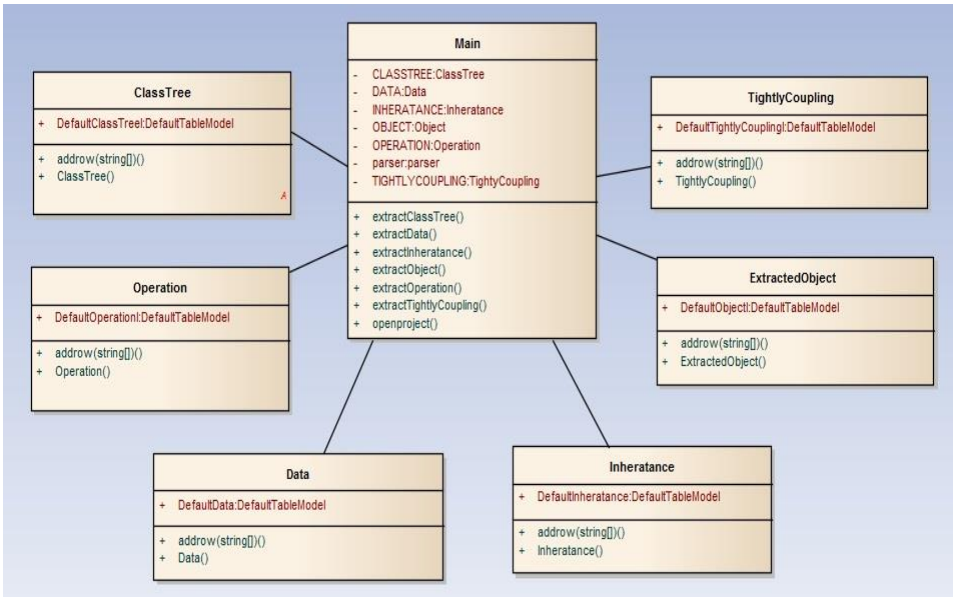

Figure 3. Class Diagram to CDPI Tool.

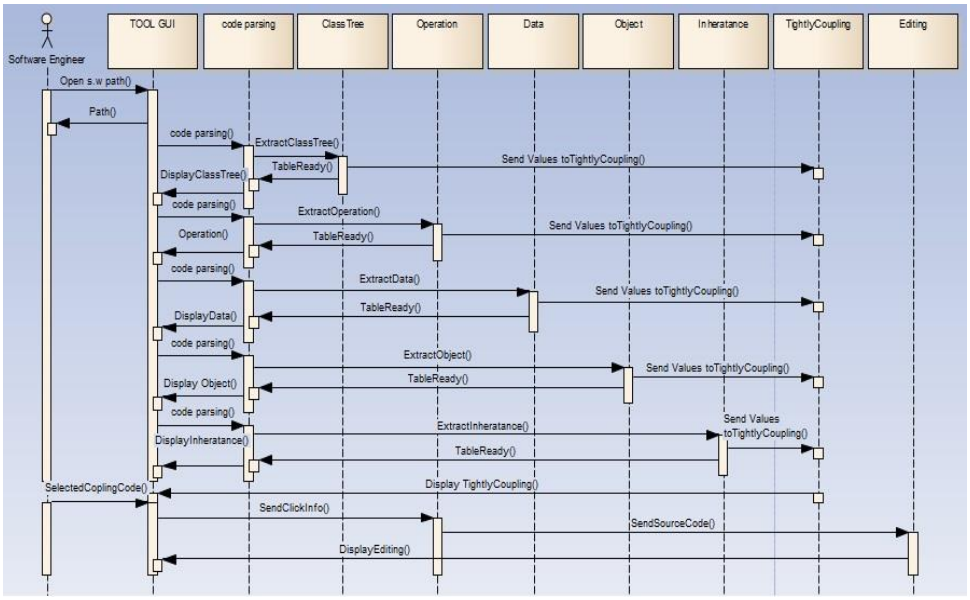

Figure 4. Sequence Diagram to CDPI Tool.

4. Case Study

This section provides a test and evaluation of the performance of the CDPI tool, as it deals with the use of the CDPI tool in a practical way with the test data (http://www.neiljohan.com/java/) and discussion the results obtained.

Technium Vol. 4, Issue 6 pp.84-98 (2022)
ISSN: 2668-778X
www.techniumscience.com

4-1 Testing of CDPI tool.

The CDPI tool was tested by software written in the object-oriented programming style of the Java language.

The CDPI tool consists of a number of buttons, starting with the button to selected the Package Path for Software, then the button to display the Classes Tree, the button to display the Operations, the button to display the Data, the button to display the Objects, the button to display Inheratance, the button to display the Tightly Coupling, and finall the button to display the Decoupling Editing . The main interface of the CDPI tool is shown in Figure (5).

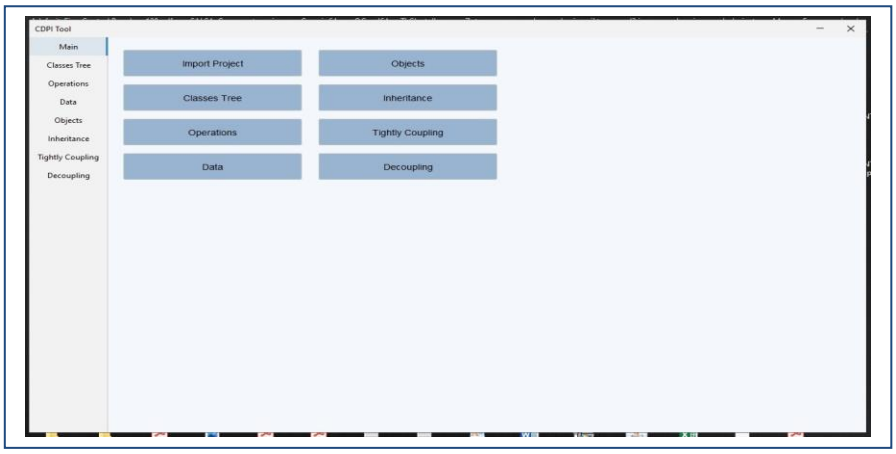

Figure 5. Main Interface to CDPI Tool.

The CDPI tool performance test results will be displayed in the decoupling process as follows:

1- Selected the Package Path for Software (test data), as shown in Figure (6).

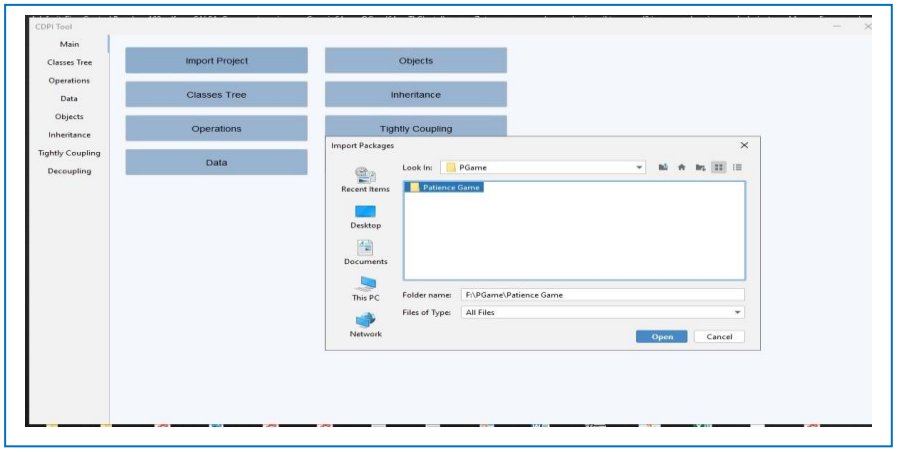

Figure 6. Import Packages.

2- Extracting Class Tree table that contain the packages name and the class/ interface of each package, as shown in Figure (7).

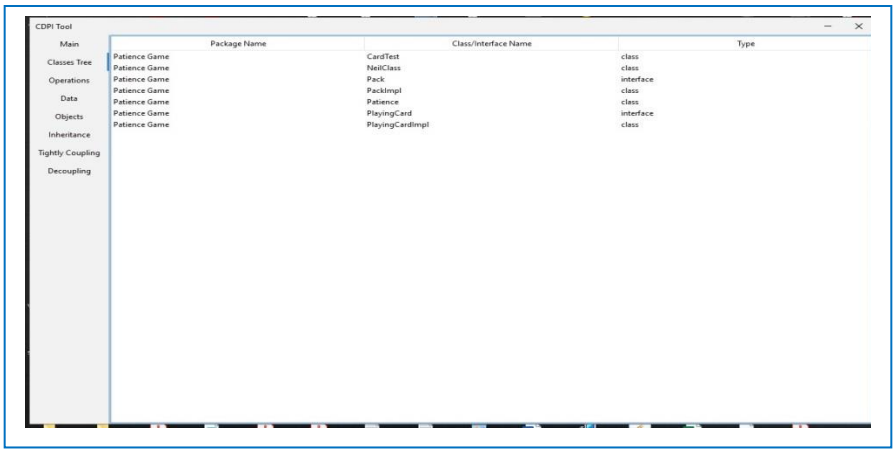

Figure 7. Class Tree table.

Class Tree Table for the entered software in Figure (7), shows that the software consists of one Package, five classes and two interfaces. The Class Tree Table gives the user a vision to the software components without need to go back to the software itself and analyze it manually.

3- Obtaining Operations Information for each class of software through Operations Table, as shown in Figure (8).

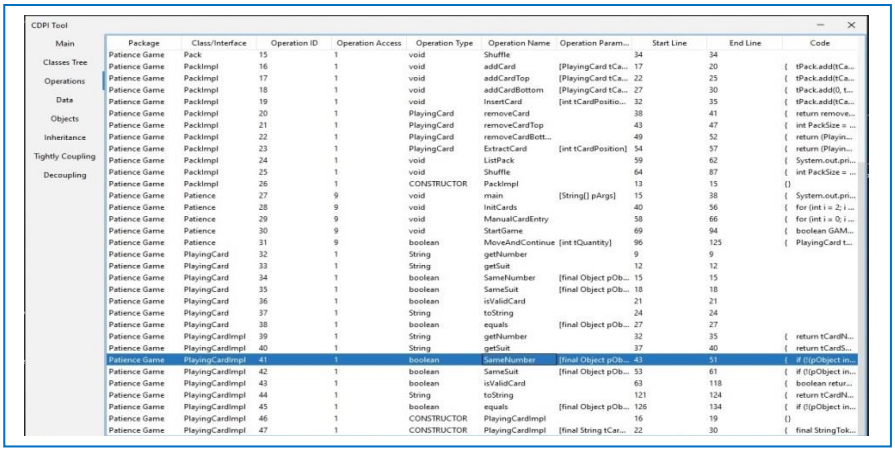

Figure 8. Operations Table.

4- Obtaining detailed information about Global and Local variables for each class of software through Data Table in Figure (9).

Technium Vol. 4, Issue 6 pp.84-98 (2022)
ISSN: 2668-778X
www.techniumscience.com

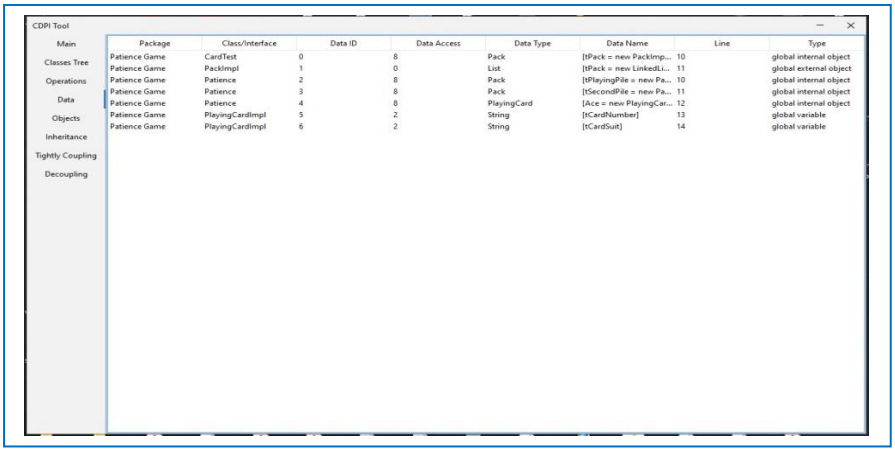

Figure 9. Data Table.

5- Displaying the dependency of the class on other class through Objects Table shown in Figure (10).

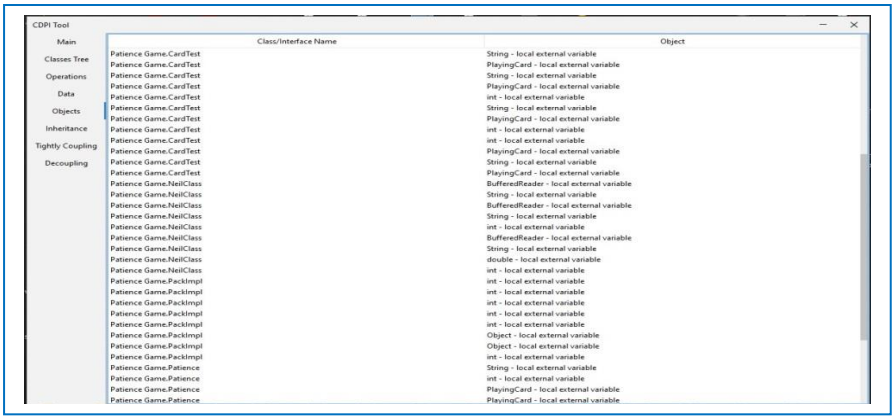

Figure 10. Objects Table.

6- Clarify the relationships of the software classes and interfaces through Inheratance Table, as shown in Figure (11).

7- Detect coupling among classes of object oriented Java softwar and displays them in Tightly Coupling table, as shown in Figure (12).

Technium Vol. 4, Issue 6 pp.84-98 (2022)
ISSN: 2668-778X
www.techniumscience.com

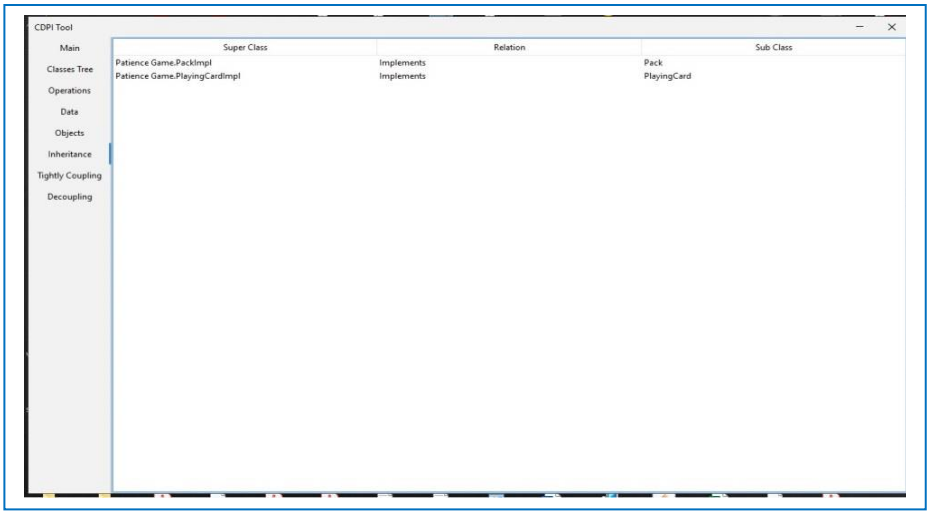

Figure 11. Inheratance Table.

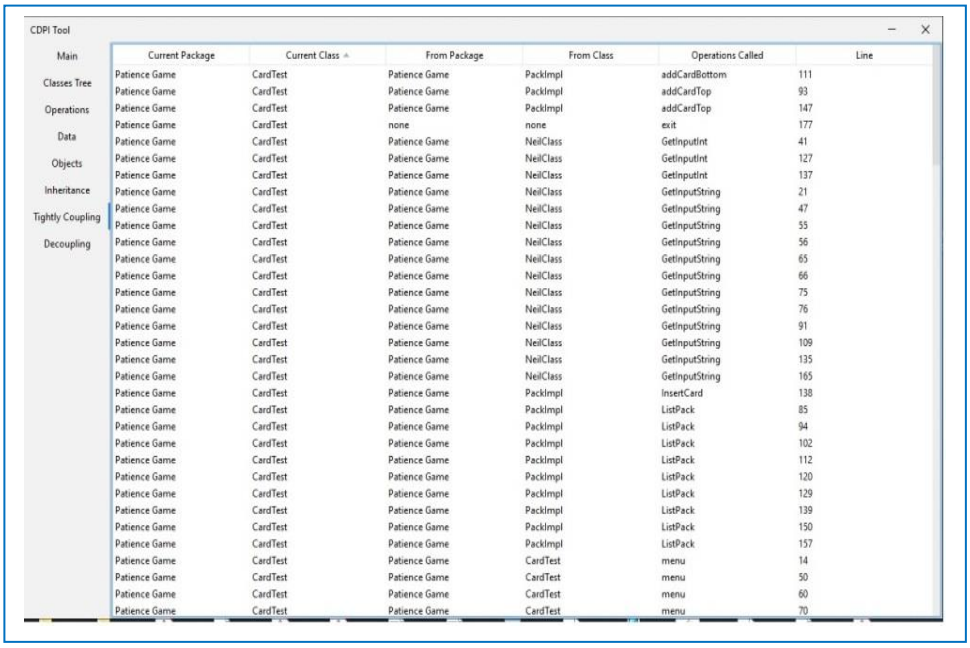

Figure 12. Tightly Coupling Table.

8- Displaying source code befor and after Decoupling through Decoupling Editing, as shown in Figure (13).

Technium Vol. 4, Issue 6 pp.84-98 (2022)
ISSN: 2668-778X
www.techniumscience.com

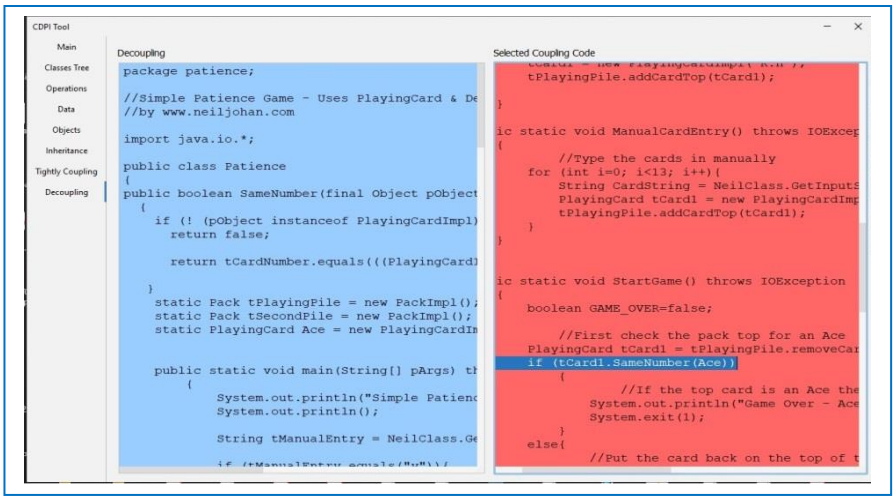

Figure 13. Decoupling Editing.

The following is a discussion of the results obtained from the Testing of CDPI tool.

We note from Tightly Coupling table Figure (12) that the CDPI tool candidating the coupling that inside the test data to the software engineer, who in turn makes an accurate selection of the coupling that he wants to decoupling after noticing the tables generated by the tool.

Figure (14) shows that the Method **SameNumber** has been called twice by the Class **Patience** only.

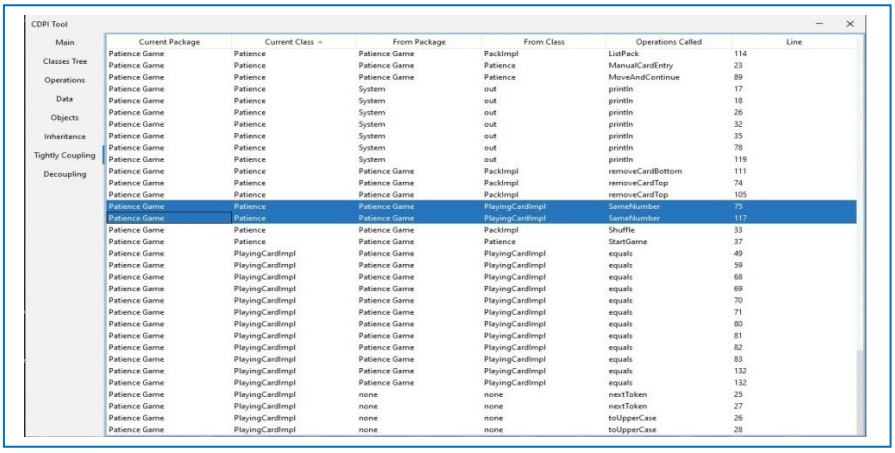

Figure 14. Coupling selection.

When the software engineer clicks on it, the Object **tcard1** of the **playingcard** type as shown in Figure (13) will be canceled and the code for the Method **SameNumber** will be fetched from the Operation table as shown earlier in the figure (8), and the Method **SameNumber** is transferred from the Class **PlayingCardImpl** to the Class **Patience** as shown in the figure (15) and figure (16).

Technium Vol. 4, Issue 6 pp.84-98 (2022)
ISSN: 2668-778X
www.techniumscience.com

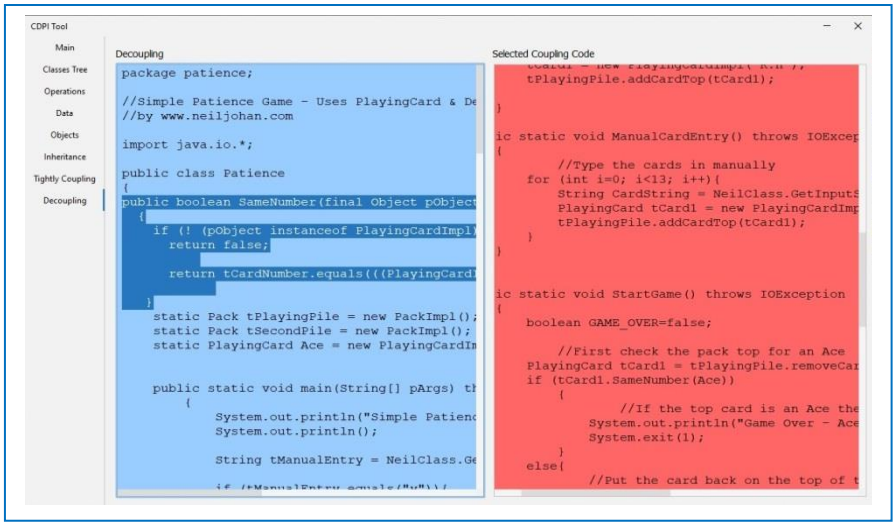

Figure 15. Method transfer.

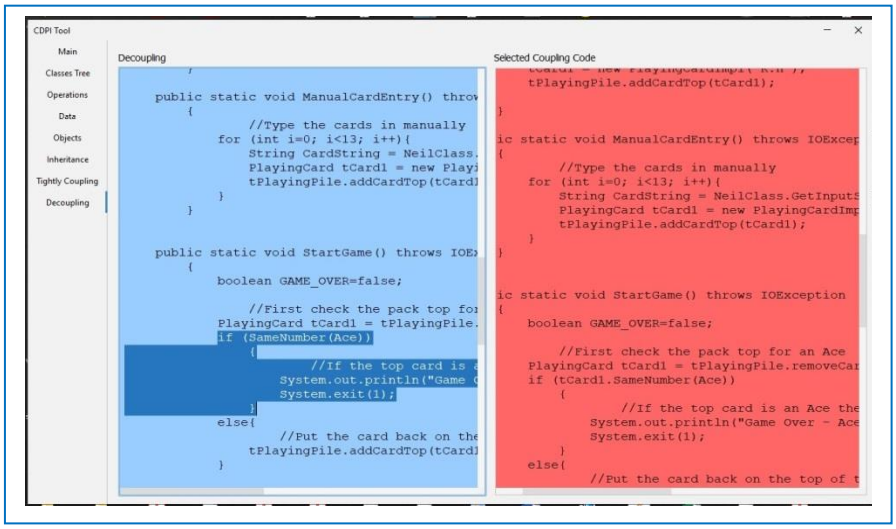

Figure 16. Object cancele.

4-2 Evaluating of CDPI tool.

In order to evaluate the performance of the CDIP tool, the Time Copmlexity (Big O notation [11]) calculates to the source code for a selected coupling by software engineering before and after decoupling it , To see the effect of reducing coupling on Optimize Runtime Performance of the software. Figure (17) shows the calculation of Time Complexity to Method **SameNumber** before decoupling.

Technium Vol. 4, Issue 6 pp.84-98 (2022)
ISSN: 2668-778X
www.techniumscience.com

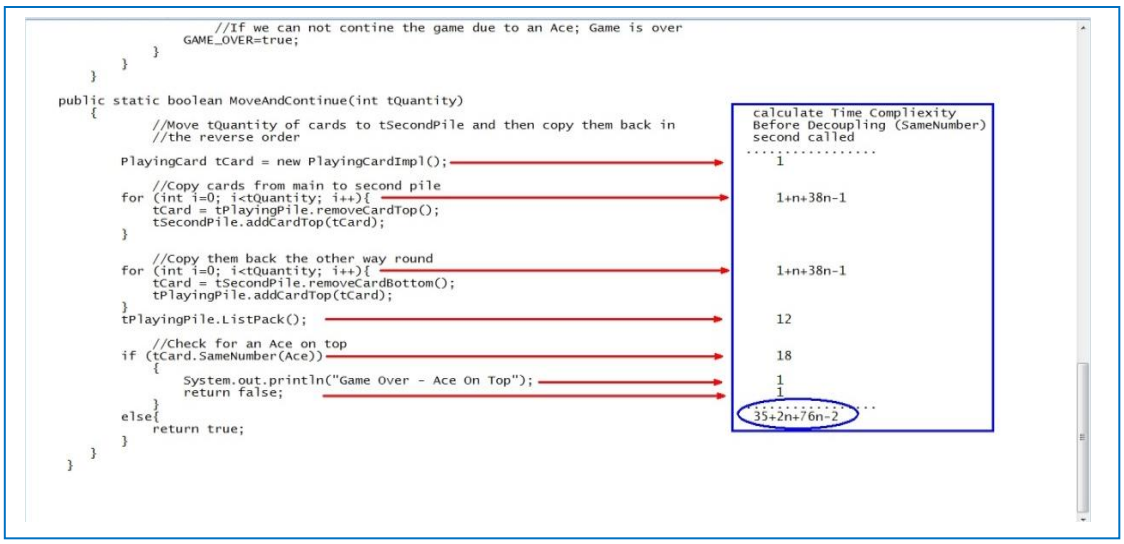

Figure 17. Calculates Time Copmlexity before decoupling.

From Figure (17) we can see that the total number of time required to complete execution before decoupling the Method **SameNumber** is :

$$F(n)= 35+2n+76n-1 \quad (1)$$

Figure (18) shows the calculation of Time Complexity after decoupling the Method **SameNumber,** Note that the Class **Patience** has called the Method twice, here we will calculate the second called only.

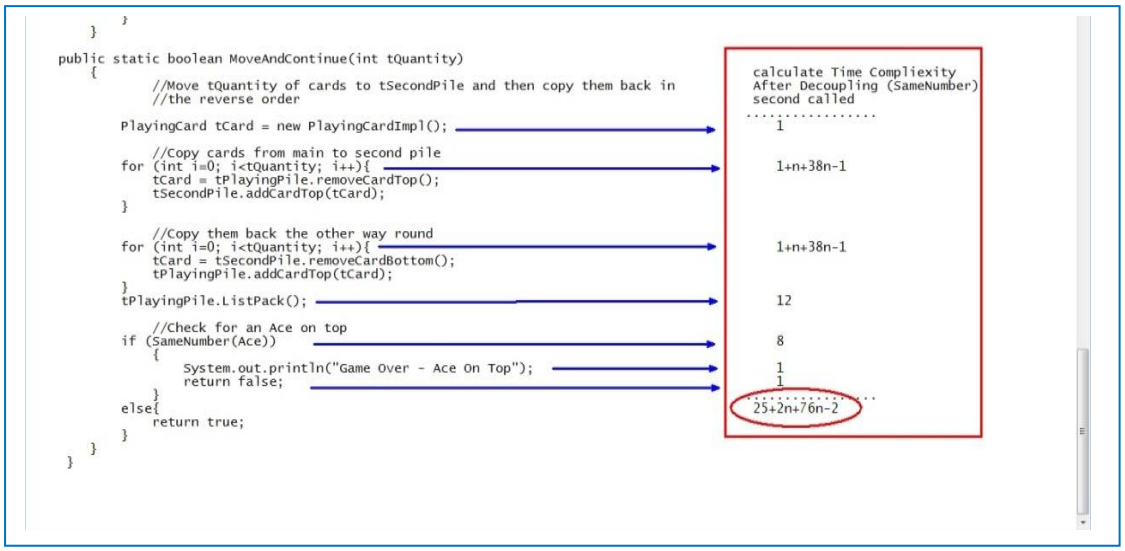

Figure 18. Calculates Time Copmlexity after decoupling.

Technium Vol. 4, Issue 6 pp.84-98 (2022)
ISSN: 2668-778X
www.techniumscience.com

From Figure (18) we can see that the total number of time required to complete execution after decoupling the Method **SameNumber** is :

$$F(n)= 25+2n+76n-1 \quad (2)$$

We notice from equations (2) that the source code after the decoupling is executed faster than before decoupling to produce the required output, which is an indication of Optimize Runtime Performance of the program.

5. Conclusion and Future Work

In this paper, It was noticed that the software engineering tools that were studied and analyzed detect coupling without trying to do decoupling. So, A CASE Tool has been proposed which is called CDPI (Coupling Detection to Performance Improvement). Which discovers the coupling and presents it to the software engineer with the possibility of decoupling any coupling wanted in order to reduce the impact of coupling on Runtime Performance .

5-1 A number of conclusions were recorded during the construction and testing of the CDPI tool performance:

1- The tables generated by the CDPI tool simulates the work of the class diagram of showing relationships and dependencies and provides other information about the internal details that bind software parts, and results in an adequate understanding of the software engineer .

2- The software engineer must make the decision to choose the decoupling of the candidate coupling and not to decoupling automatically by the tool.

3- Reducing the degree of dependency between the model (decoupling), which is an indication of Optimize Runtime Performance of the program.

5-2 In order to develop the CDPI tool and increase the percentage of beneficiaries, the following has been suggested:

1- The CDPI tool should have the ability to deal with software written in other languages.

2- Integrating machine learning with the CDPI tool to to make the decision to decouple, Without intervention of a software engineer

6. Acknowledgment

The authors would like to thank the University of Mosul/ College of Computer Sciences and Mathematics for their provided facilities, which helped to improve the quality of this work.

7. References

[1] Ajienka, N., Capiluppi, A. (2017). "Understanding the Interplay between the Logical and Structural Coupling of Software Classes". Journal of Systems and Software, 134, 120-137.
[2] Altaie, A. (2022). "Designing and implementing a tool for measuring cohesion and coupling of Object-Oriented Systems". Turkish Journal of Computer and Mathematics Education, Vol.13 No.02 (2022), 368-375
[3] Alzamil, Z.A. (2018). "Software Components' Coupling Detection for Software Reusability". International Journal of Advanced Computer Science and Applications(IJACSA),9(10).
[4] Anwer, S., Adbellatif, A., Alshayeb, M., and Anjum, M. S., (2017). "Effect of coupling on software faults: An empirical study". International Conference on Communication, Computing and Digital Systems (C-CODE), 2017, pp. 211-215.
[5] Apul, S. (2021). "Bimar Software Quality Portal: Experience and Lessons Learned" 15th Turkish National Software Engineering Symposium (UYMS), pp. 1-3.
[6] Bidve, V.S., Sarasu, P., Pathan, S.K., & Pakle, G.K. (2019). "Web Based Tool for Measuring Coupling in Object-Oriented Software Modules". International Journal of Intelligent Engineering and Systems. Vol.12, No.4, 2019.
[7] Bidve, V.S., Sarasu,P. (2016). "Tool for Measuring Coupling in Object- Oriented Java Software". International Journal of Engineering and Technology (IJET), Vol 8 No 2 Apr-May 2016.
[8] Czibula, I., Oneţ-Marian, Z., Vida, R.F. (2019). "Automatic Algorithmic Complexity Determination Using Dynamic Program Analysis". In Proceedings of the 14th International Conference on Software Technologies (ICSOFT 2019). SCITEPRESS - Science and Technology Publications, Lda, Setubal, PRT, 186–193.
[9] Fregnan, E., Baum, T., Palomba, F., Bacchelli, A. (2019). "A survey on software coupling relations and tools". Information and Software Technology. 2019, 107,159–178.
[10] Gethers, M., Poshyvanyk, D.(2010). "Using Relational Topic Models to capture coupling among classes in object-oriented software systems". IEEE International Conference on Software Maintenance, 2010, pp. 1-10.
[11] Goodrich, M.T. Tamassia, R. (2009). "Algorithm Design: Foundations, Analysis and Internet Examples". (2nd. ed.). John Wiley & Sons, Inc., USA.
[12] Li, H., Wang, T., et al. (2021). "Mining key classes in java projects by examining a very small number of classes: a complex network-based approach". IEEE Access, vol. 9, pp. 28076–28088.
[13] Miholca, D-L., Czibula, G., Tomescu, V. (2020). "COMET: A conceptual coupling based metrics suite for software defect prediction". Proc. Comput.Sci., vol. 176, pp. 31–40.
[14] Miholca, D-L., Oneţ-Marian, Z. (2020). "An analysis of aggregated coupling's suitability for software defect prediction". 22nd International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), 2020, pp. 141-148.
[15] Paixao, M., Harman, M., Zhang, Y., Yu, Y. (2018). "An Empirical Study of Cohesion and Coupling: Balancing Optimization and Disruption". in IEEE Transactions on Evolutionary Computation, vol. 22, no. 3, pp. 394-414.
[16] Papadopoulos L, Marantos C, et al. (2018). "Interrelations between software quality metrics, performance and energy consumption in embedded applications". In: Proceedings of the 21st International Workshop on Software and Compilers for Embedded Systems, Association for Computing Machinery, New York, NY, USA, SCOPES '18, p 62-65
[17] Saeed, M.G., Paolone, G., Di Felice, P. (2022). "Hierarchical Evaluation of Software Projects: An Experiment". In: Arai, K. (eds) Proceedings of the Future Technologies Conference (FTC) 2021, Volume 3. FTC 2021. Lecture Notes in Networks and Systems, vol 360. Springer.

[18] Saxena, V., Kumar, S. (2012). "Impact of Coupling and Cohesion in Object-Oriented Technology, Journal of Software Engineering and Applications". Vol. 5 No. 9, Pp. 671-676.
[19] Saydemir, A., Simitcioglu, M. E., Sozer, H.(2021). "On the Use of Evolutionary Coupling for Software Architecture Recovery". 15th Turkish National Software Engineering Symposium (UYMS), 2021, pp. 1-6.
[20] Seiffert, C., Khoshgoftaar, T. M., Van Hulse, J., Folleco, A. (2014). "An Empirical Study of the Classification Performance of Learners on Imbalanced and Noisy Software Quality Data," Information Sciences. Vol 259 (February, 2014), 571–595.
[21] Singh, V., Bhattacherjee, V. (2013). "Identifying Coupling Metrics and Impact on Software Quality". International Journal of Engineering and Technology (IJET), Vol 5 No 4.
[22] Sousa, B.L., Bigonha, M.A., Ferreira, K.A. (2019). "Analysis of Coupling Evolution on Open Source Systems". In Proceedings of the XIII Brazilian Symposium on Software Components, Architectures, and Reuse, Salvador, Brazil, 23–27 September 2019; pp. 23–32.
[23] Vaz, R., Shah, V., Sawhney, A., Deolekar, R. (2017). "Automated Big-O analysis of algorithms" International Conference on Nascent Technologies in Engineering (ICNTE), pp. 1-6.
[24] Woodside, M., Franks, G., Petriu, D. C. (2007). "The Future of Software Performance Engineering". Future of Software Engineering (FOSE '07), pp. 171-187.
[25] Yu, L., Ramaswamy, S. (2011). "Examining the Relationships between Software Coupling and Software Performance: A Cross-platform Experiment". Journal of computing and information technology, 19 (1), 1-10.