International Journal of Recent Technology and Engineering (IJRTE)
ISSN: 2277-3878 (Online), Volume-7 Issue-6, March 2019

PLC Reducer – A Tool to Generate Possible Reductions in Coupling at Design Level

Aprna Tripathi, Rahul Pradhan, Ankur Chaturvedi

Abstract: (ABS) Coupling and cohesion are the two prime attribute that define the quality of a software design. High coupling is an undesirable feature while higher cohesion is enviable property. A number of software quality parameters like maintainability, readability, understandability etc. are directly or indirectly related with coupling and cohesion and thus it become necessary to pay a great attention towards desirable degrees of coupling and cohesion during design phase of software development life cycle. In this paper, an algorithm PLC Reducer is proposed that suggested the possibilities how the coupling can be reduced in a design and also generates a redesign for the designed software. A complete demonstration of algorithm functionality is shown for a project. Also, algorithm is applied on five different java-based projects and the amount of coupling before and after applying the algorithm is shown in the paper.

Keywords: (ABS) Cohesion, Coupling, PLC, PLC Reducer

I. INTRODUCTION

In the object-oriented software development [1, 2], coupling and cohesion are among the key metrics to measure the quality of software. According to Stevens et al. [3], coupling is "the measure of the strength of association established by a connection between two modules." Therefore, the stronger the coupling between modules, the more integrated these are, the more difficult these modules are to understand, change, and maintain and thus the finally software system become more complex. Whereas, a module has strong cohesion if it represents exactly one task of the problem domain and all its elements contribute to this single task [4]. High coupling is an undesirable feature while high cohesion is a demanding property of the software. A number of software quality attributes like maintainability, understandability, reusability [5, 6, and 7] are directly or indirectly depends upon the degree of coupling and cohesion. One of the leading benefits of low coupling is high reusability [8, 9, and 10]. Coupling and reusability of any software are inversely related. To enrich the reusability, only measuring the coupling of software is not enough, until it is reduced if there is any possibility at all. This design phase activity shall lead to software development that is maintainable requiring lesser testing effort and understandability.

Revised Manuscript Received on December 22, 2019.
* Correspondence Author
Aprna Tripathi, Department of Computer Engineering and Applications, GLA University, Mathura (U.P), India.
Rahul Pradhan, Department of Computer Engineering and Applications, GLA University, Mathura (U.P), India.
Ankur Chaturvedi, Department of Computer Engineering and Applications, GLA University, Mathura (U.P), India.

Most of the authors focus on removing the cyclic dependency while reducing the coupling of the component [11, 12, and 13]. Since the quality of software depends on both cohesion and coupling, thus during reduction of coupling, cohesion can't be overlooked. It is imperative to give equal focus on coupling as well as cohesion in the process of coupling reduction [14]. Any compromise with the cohesion strength impacts the quality of the software.

Since the quality of software depends on both cohesion and coupling, thus during reduction of coupling, cohesion can't be overlooked. It is imperative to give equal focus on coupling as well as cohesion in the process of coupling reduction [14]. Any compromise with the cohesion strength impacts the quality of the software.

The mechanism to reduce the coupling could be in the form of some change in logic or shuffle of the classes existing in various packages. This paper proposes an algorithm 'PLC Reducer' to reduce the coupling. It generates few suggestions that include the shifting of methods from a class of one package to the class of other package. When the suggested changes are incorporated, a fall in the package level coupling is observed. This is because of reduction of sub-connection. Thus, these suggestions are used by the proposed approach to restructure the design and a significant reduction in the coupling is observed. The decision to shift a method from one package to another is based on two issues: When to shift the method and where to shift the method.

$$Weight\ of\ Sub\_Connection\ W_{P_iC_A-P_jC_B} = \frac{Number\ of\ methods\ called\ by\ class\ A}{Number\ of\ methods\ defined\ in\ class\ B} = \frac{M_c}{M_d} \quad \text{.....(1)}$$

$$Weight\ of\ Connection(P_{ij}) = \frac{Number\ of\ classes\ used}{Number\ of\ classes\ defined} \sum_{j=1}^{N} weight\ of\ Sub\_Connections\ from\ P_j\ to\ P_i\ for\ j \neq i \quad \text{.....(2)}$$

where, N is the number of sub-connection, P represents the set of packages P = {P_1, P_2, P_3 ....., P_i,.... P_j..... P_n}, and P_i stands for the calling package, and P_j signifies the called package and $P_j \in \{P - P_i\}$

$$PLC = \frac{Number\ of\ package\ used}{Total\ number\ of\ package - 1} \times (sum\ of\ nubmer\ of\ connection\ and\ sub-connection) \times \sum_{j=1}^{N} weight\ of\ Connections\ from\ P_j\ to\ P_i \quad \text{.....(3)}$$

A. When to Shift a Method

For a particular method, the number of calls for this method from other classes of the same package as well as from other packages is counted. The number of calls of a method in the same package is considered as internal call while the number of calls for a method from other packages is considered as external call. In case when the count of internal call is less than that of external call, the method to be shifted is identified.

B. Where to Shift the Method

After taking the decision to shift a method, we shift the method into a class of the package that has maximum external call value. To find the class where we have to move this method, external call value is analyzed. Since, external call is the sum of calls of the method through different classes of the package. Thus, the class that has maximum value of call is selected for shifting this method.

C. Brief about PLC and PLCoh

To observe the impact of PLC Reducer, a package level coupling metric PLC [15] and cohesion metric PLCoh [ ] is considered for experiment. To understand the PLC and PLCoh, there is a need to understand the terminology used to compute the PLC and PLCoh.

C1. Connection

There may be multiple package exists and one package may be completely used by other package or partially used by other package. When there is some content of one package is used by other package, a link is established and such link is named as connections. Package is considered as a basic reusable by the software developers. Here, a connection is the representation of the abstract relationship to show the coupling between two packages. Thus, connection represents the number of other package this package is dependent upon.

C2. Sub-Connection

In order to establish a relationship between two packages, the only way is the relation between classes defined inside these packages. Similar to the definition of connection, when class-A of one package uses any method existing in class-B of the other package, there exists a link from class-A to class-B. This link is called sub- connection. Sub-connections are actual relationships that are established between classes of different packages to reuse the functionalities. Thus sub-connections aggregate to form connection.

C3. Sub-Connection and Its Weight

A package consists multiple classes and there may be multiple methods in a class. Then either a sub-connection may be established due to single call of method or multiple methods may be used. In order to differentiate these two situations, weight for each sub-connection is considered in the proposed work. Following is the process to assign the weight to these sub-connections:

If there exist Md number of methods in Class-B (C_B) of package P2 and Class-A (C_A) of package P1 calls Mc out of Md number of methods, then the weight of this sub-connection is defined as shown in equation 1.

C4. Weight of Connection

There will be existence of connection between any two packages if at least one sub-connection among classes of two different packages exists. The weight of connection depends upon two parameters- number of classes used and the weight of the sub - connection. Thus, weight of connection is the multiplication of ratio of class used to the total number of classes of a package and sum of weights of all the sub-connections that exist between two packages. Mathematical expression is shown in equation 2.

C5. Direct Relationship: A relationship between two classes is said to be direct, i.e. a class directly uses the methods of other class. As shown in figure1, class C1 directly uses the method M2 that is defined in class C2.

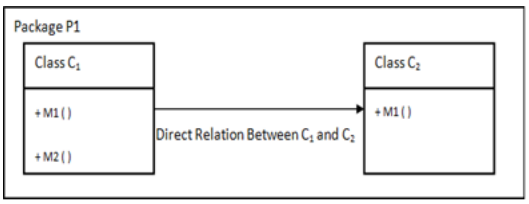

Figure 1. Example of Direct Relationship

C6. Transitive Relationship: When a class C_1 uses the method of other class C_2 that again calls other classes C3 to complete the functionality. The relation between classes C1 to class C_3 is known as transitive relationship. In figure 2, method M_3, defined in class C_2 is used in class C_1, while method M3 requires a value M that is returned by method M_6 defined in class C_3. Thus, the relationship between class C_1 and C_3 is an indirect / transitive in nature.

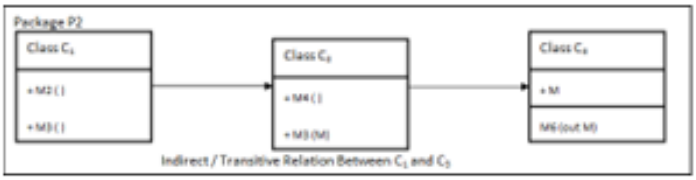

Figure 1. Example of Indirect / Transitive Relationship

C7. PLC Formulation

Here, we assumed that P is the set of packages in the software and the package for which coupling has to be calculated is Pi. If

N =total number of packages in the project and

P= set of packages then P= {P1, P2, P3, Pi.....PN}.

To compute coupling for Pi, we examined the connections between the package Pi and packages {P-Pi} along with their weight. Mathematical expression is shown in equation 3.

$$PLC = \frac{Number\ of\ package\ used}{Total\ number\ of\ package - 1} \times (sum\ of\ number\ of\ connection\ and\ sub-connection) \times \sum_{j=1}^{N} weight\ of\ Connections\ from\ P_j\ to\ P_i \quad \text{.....(3)}$$

C8. PL Coh Formulation

Let us assume a package P and in which a class named as C_i exists then

Ri = number of relationships (direct or transitive) related with class C_i

N = total number of classes in package P

Then,

Cohesion of Class C_i is expressed as

$$Cohesion\ of\ Class\ C_i\ (C_iCoh) = \frac{Number\ of\ Relationships\ (directly\ and\ / or\ Transitively\ Related\ With\ C_i)}{(N-1)} \quad \text{.....(4)}$$

For coupling itself relation it not possible thus number of classes is considered as N-1 in calculation. PLCoh of package P is normalized value of C_i Coh for i = 1 to N. Mathematically PLCoh is shown in eq.5

$$Package\ Level\ Cohesion\ (PLCoh) = \frac{\sum_{i=1}^{N} C_i Coh}{N} \quad \cdots\cdots\cdots \quad (5)$$

The paper is dived into four sections. Section 2 presents the state of art related with coupling reduction. The proposed algorithm PLC Reducer is discussed in section3. In section 4 the impact on PLC after applying the suggestions is analyzed. The proposed work is concluded in conclusion section.

II. SATE OF ART

This section presents a survey of leading papers related to the coupling reduction approaches.

Martin [16] stated that coupling is desirable, because if we ban coupling between modules, we have to put everything in one big module. Thus, coupling can only be controlled. It can be controlled through avoiding the cyclic dependencies between the modules. To reduce the coupling, author visualizes the high-level dependencies and then rationalizes them by separating the interface and implementation in order to break dependencies.

An approach for automatically optimizing existing software modularizations by minimizing connectivity among packages is presented in Abdeen et al. [17]. In this paper cyclic - connectivity is based on simulated annealing which is a local search-based technique. Hautus [18] propose a tool namely Package Structure Analysis Tool (PASTA) to analyze the modular structure of java programs that is based on two aspects. Firstly it confirms the Acyclic Dependency Principle (ADP) to check cycles in the dependency graph. Secondly, if ADP is confirmed, it restructures the packages using layering concept i.e. the layer of a package is the maximum length of a dependency path to a package with no dependency.

A method to restructure a poorly structured module was proposed by Kim and Kwon [19]. They applied program slicing to extract tightly coupled sub - modules (processing blocks), and uses module strength as a criterion to identify multi - function modules to decide how to restructure such modules. Module strength is defined in terms of the level of sharing between processing blocks which was based on the code implementation,

Most authors focus on removing the cyclic dependency while reducing the coupling of the component. Since the quality of software depends on both cohesion and coupling, thus during coupling reduction phase, cohesion can't be overlooked. It is needed in the process of coupling reduction [18]. Any compromise with the cohesion strength impacts the quality of the software if only focusing on the coupling of the entire system.

III. PLC REDUCER- THE PROPOSED ALGORITHM

To reduce the design and make a package more reusable, an algorithm 'PLC Reducer' is developed that generates the possible suggestion of reducing the coupling as well as cohesion increments possibilities. PLC Reducer works on the following principle.

Shift the class that is less cohesive and participates in incrementing the package coupling. In other words, the class that is less related in the package where it is defined while highly demanded by the other packages is a candidate for shift. Thus to reduce the coupling, such classes are shifted into those package where they are highly called.

PLC Reducer generates the suggestions on the basis of coupling and cohesion. The package coupling occurs when a method of one package is used by other package and the cohesion exists when the classes of a package are related with other by using the methods of one another. To reduce the coupling, those methods are identified which are the major cause of higher coupling while within the package where these are defined are less cohesive. PLC Reducer suggests that to reduce the coupling, such method may be shifted in the class where they are more used rather than the package where it is defined. Figure 3 describe the algorithm to reduce the PLC of the existing system by shifting the methods from one class to another.

```
ALGORITHM 1. PLC Reducer
Input: Java Project PR with multiple packages
Output: Suggestions to reduce PLC.
PLCReducer(PR){
for each package P in PR do
        for each class C in package P do
                for each method M in class C do
                 OTHERPKG=maxCalls(M, PR)
                        if (OTHERPKG != P )
                        then
                        cls =FindClass( M, OTHERPKG)
                        Print suggestion "Shift  method M  from package P to package OTHERPKG
                        Print suggestion "Shift method M from class C to class cls
                        end
                end
        end
end
}
Function  maxCalls(method M, Project PR) {
 max = 0
pkg = null
for each package P in PR do
 called = numCallsinPackage(M, P)
        if (called > max)
        then
         max = called
         pkg = P
         end
end
return (pkg)
}
Function  numCallsinPackage(method M, package P){
 count = 0
for each class C in P do
 count = count + getTimesCalled(M, C)
end
return count
}
Function FindClass(Method M, Package OTHERPKG)
{
for each Class C in Package OTHERPKG
{MethodCalledinClass[i]=getTimesCalled(M, C) }
max=0
for each Class C in Package OTHERPKG
{
if (  MethodCalledinClass[i] >max)
then
max =   MethodCalledinClass[i]
cls = ClassNameofClassIndex[i]
end
end
}
return cls
}
```

Figure 3. Proposed Algorithm 'PLC Reducer'

IV. IMPACT OF PLC REDUCER ON PLC AFTER EMPLOYING SUGGESTIONS

To analyze the impact on PLC after applying the suggestions, four java based projects are considered. The detailed structure of projects is shown in figure 4. To explain the impact of shifting the methods as suggested by PLC Reducer, a project 'Javaopearation' which is one among the four projects is explored in detail. It has three packages: 'Parser', 'Source Code' and 'Test Case' and 19 classes. 4 shows the relationship among packages of 'Java operation' project.

Suggestions generated by PLC Reducer algorithm assist in identifying any reduction of the PLC for the existing system. The format of the suggestions for project 'Java operation' is shown in 5.

After employing the suggestions, few relationships between packages either disappear or get added and the same is shown in the 6.

Figure 7 demonstrates the impact of applying suggestions that is the outcome of algorithm 'PLC Reducer' on the values computed for PLC. Modified PLC (MPLC) is the coupling observed after the changes suggested by PLC Reducer are incorporated in the project. The same procedure is carried out for other four projects Lamistra, Anagram Game, Anagram Game _Modified and Shipment.

Figure 7 shows the PLC for packages of project as well as average value of PLC & MPLC of the project after employing the suggestions generated by the algorithm 'PLC Reducer'. From figure Error! Reference source not found.7 and 8, results establish that there a significant change in PLC and the average reduction in PLC is 46.86 %, thus validating the motive of 'PLC Reducer' algorithm.

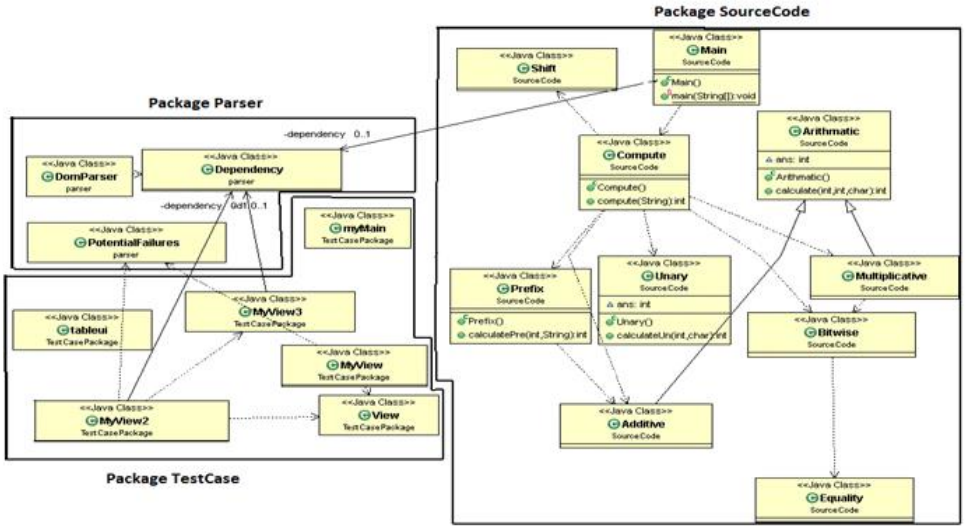

Figure 4. Package Diagram for Project - 'Java operation'

Suggestions to reduce coupling:

Shift getClasses from Class Dependency to Class View

Shift generateDependencies from Class Dependency to Class View

Shift doBFS from Class Dependency to Class View

Shift getClassArray from Class Dependency to Class MyView2

Shift getMaxId from Class Dependency to Class View

Shift add from Class Additive to Class Dependency

Shift getClasses from Package parser to Package TestCasePackage

Shift generateDependencies from Package parser to Package TestCasePackage

Shift doBFS from Package parser to Package TestCasePackage

Shift getClassArray from Package parser to Package TestCasePackage

Shift getMaxId from Package parser to Package TestCasePackage

Shift add from Package SourceCode to Package parser

Figure 5. Output of Algorithm ' PLC Reducer ' for Project 'Java operation'

Table 1. Summary of PLC and MPLC for Projects

Project	LOC	Package	PLC	MPLC	Project Average		No. of Sub - connections		% reduction in PLC
					PLC	MPLC	Before	After	
Java operation	899	Test Case	1.33	0.4	0.55	0.2	4	2	63.60%
		Parser	0.33	0			1	0	
		Source Code	0	0.2			0	1	
Lamistra	4932	Editor	0.39	0.28	0.72	0.52	10	8	27.70%
		Stratgo	0.25	0.19			3	2	
		Server	1.15	1.7			22	28	
		Player	1.8	1.6			36	31	
		Remote	0	0			0	0	
Anagram Game	395	Lib	0	0	0.3	0.25	0	0	17%
		Ui	0.6	0.5			1	1	
Anagram Game _Modified	402	Lib	0.5	0.2	0.25	0.1	0	0	60%
		Ui	0	0			1	1	
Shipment	792	Shipment	0.4	0	0.73	0.25	1	0	65.75%
		Shipment Company Detail	1.06	0.5			4	1	

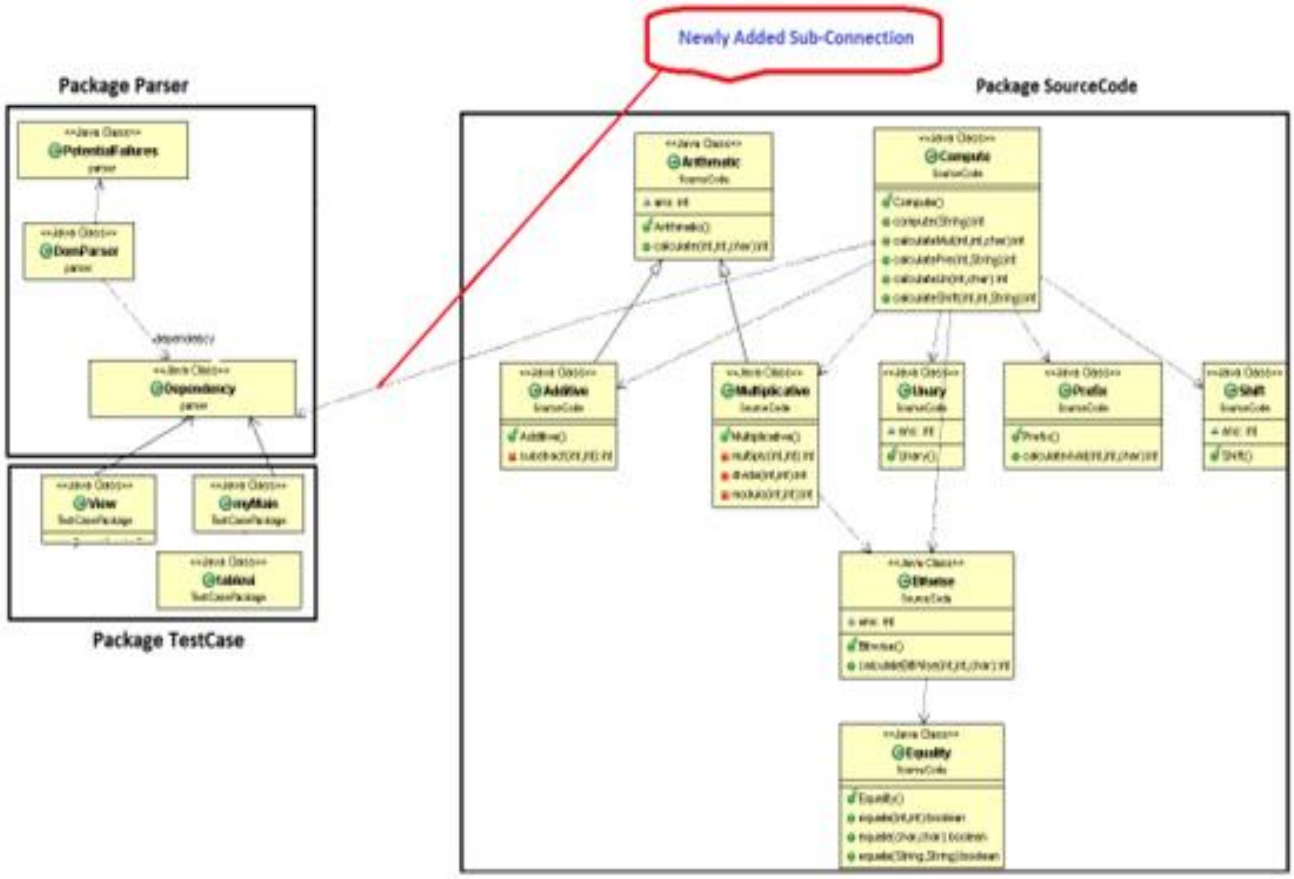

Figure 6. Package Diagram for Project - 'Javaoperation' after Employing Suggestions

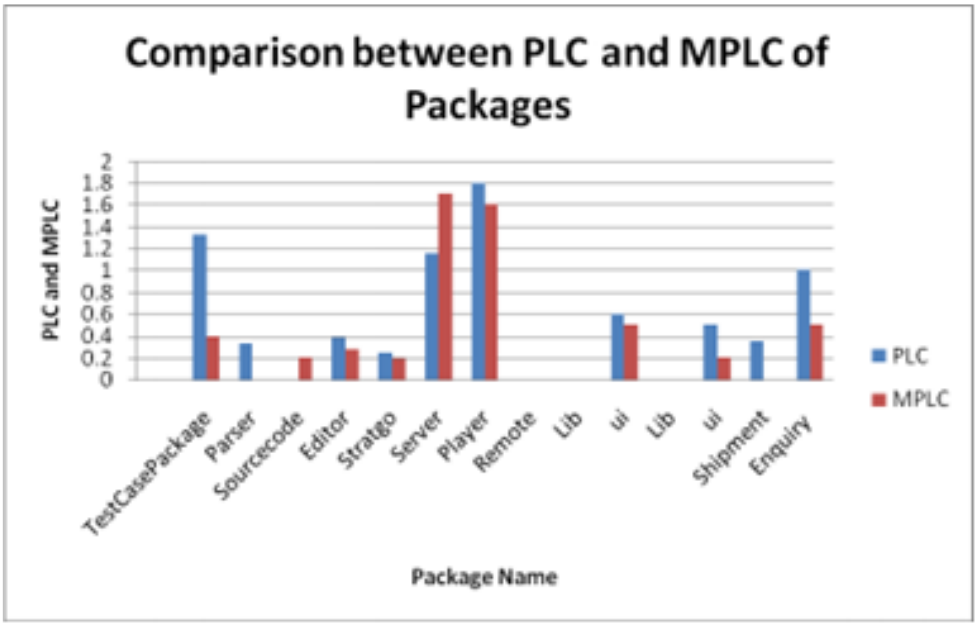

Figure 7. PLC and MPLC for Various Packages of Six Projects

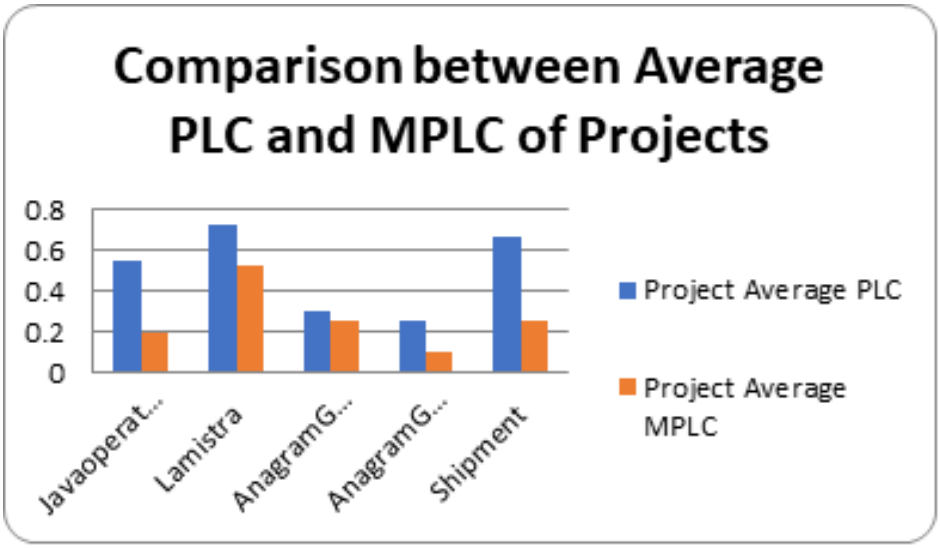

Figure 8. Average PLC & MPLC of Projects

V. CONCLUSIONS

This paper proposes an algorithm 'PLC Reducer' that is used to reduce the PLC in case of faulty design. The package coupling occurs when a method of one package is used by other package and the cohesion exists when the classes of a package are related with other by using the methods of one another. To reduce the coupling, those methods are identified which are not cohesive within the package and are used by other package. This is because of reduction of sub-connection. The proposed PLC Reducer algorithm suggests that to reduce the coupling, such method may be shifted in the class where these are more used rather than the package where it is defined.

To validate the objective of the 'PLC Reducer' algorithm, four java projects are considered. After employing the suggestions generated by the algorithm 'PLC Reducer', the result shows that there a significant change in PLC and the average reduction in PLC is about 46.86 %. Also the impact of PLC Reducer on PLCoh is analyzed and it is found that either PLCoh increases or remains unchanged, thus validating the 'PLC Reducer' algorithm.

REFERENCES

1. Grady Booch. 1993. Object-Oriented Analysis and Design with Applications (2nd Ed.). Benjamin-Cummings Publ. Co., Inc., Redwood City, CA, USA.
2. Peter Wegner, Concepts and paradigms of object-oriented Programming. SIGPLAN OOPS Mess. 1, 1 (August 1990), pp. 7-87, 1990.
3. W. Stevens, G. Myers, L. Constantine, Structured Design, IBM Balagurusamy, Programming in ANSI C, Tata McGraw-Hill Education, 2008, ISBN 9780070648227
4. Systems Journal, 13 (2), pp. 115-139,1974.
5. T. Sheldon, K. Jerath, and H. Chung. Metrics for maintainability of class inheritance hierarchies. Journal of Software Maintenance and Evolution: Research and Practice, 14(3), pp. 147–160, 2002.
6. Jin-Cherng Lin; and Kuo-Chiang Wu, "A Model for Measuring Software Understandability,". CIT '06. The Sixth IEEE International Conference on Computer and Information Technology, pp.192-19, 2006.

7. Jin-Cherng Lin; and Kuo-Chiang Wu, "Evaluation of software understandability based on fuzzy matrix," IEEE International Conference on Fuzzy Systems, FUZZ-IEEE 2008. pp.887-892, 2008
8. Gui Gui and Paul D. Scott. "Ranking reusability of software components using coupling metrics", Journal System and Software, pp. 1450-1459, 2007.
9. G. Gui and P. D. Scott., "Coupling and cohesion measures for evaluation of component reusability", International workshop on Mining software repositories(MSR '06). ACM, New York, NY, USA, pp. 18-21, 2006.
10. Gui Gui; and Scott, P.D., "New Coupling and Cohesion Metrics for Evaluation of Software Component Reusability,", The 9th International Conference for Young Computer Scientists ICYCS, pp. 1181-1186, Nov. 2008
11. Juergen Rilling and Tuomas Klemola, "Identifying Comprehension Bottlenecks Using Program Slicing and Cognitive Complexity Metrics", Proceedings of the 11th IEEE International Workshop on Program Comprehension (IWPC'03), 2003
12. Klemola, T., "A cognitive model for complexity metrics", Proceedings of the 4th International Workshop on Quantitative Approaches in Object-Oriented Software Engineering, pp. 1-7, 2000.
13. Mayrhauser A., and A. M. Vans, "Program Understanding Behavior During Adaptation of Large Scale Software", Proceedings of the 6th International Workshop on Program Comprehension, IWPC '98, pp. 164-172, 1998.
14. M. Fowler. "Reducing coupling", IEEE software, 2001.
15. Tripathi, A. & Kushwaha, D.S. "A metric for package level coupling" CSIT (2015) 2: pp 217–233
16. Martin R., "Object Oriented design quality metrics: an analysis of dependencies", ROAD, 1995.
17. Hani Abdeen, Stephane Ducasse, Houari Sahraoui, and Ilham Alloui., "Automatic Package Coupling and Cycle Minimization", In Proceedings of the Working Conference on Reverse Engineering (WCRE '09). IEEE Computer Society, Washington, DC, USA, pp. 103 - 112, 2009.
18. Hautus E., "Improving Java software through package structure analysis", In Proc. International Conference on Software Engineering and Applications, Cambridge, USA, pp. 4 - 6, 2002.
19. Hyeon Soo Kim, Yong Rae Kwon, And In Sang Chung, " Restructuring programs through program slicing", International Journal of Software Engineering and Knowledge Engineering, Vol 4, pp. 349, 1994.
20. Aprna Tripathi, Manu Vardhan, and Dharmender Singh Kushwaha, " Package Level Cohesion and its Application", Fifth International Conference on Advances in Communication, Network, and Computing – CNC 2014, Elsevier, Chennai, Feb 21-22, 2014
21. M. Fowler. "Reducing coupling", IEEE software, 2001.