Impacts of PLC Reducer on Software Design Cohesiveness

Aprna Tripathi, Varsha Kumari, Nikhil Govil

Abstract: ***There is a number of software quality attributes depends on the software design complexity. Coupling and cohesion are the two main parameters through which software complexity can be measured. To achieve the high degree of maintainability, understandability, readability the high degree of cohesion is required. PLCReducer is a tool to review the software design complexity [ ]. In this paper, the impact of PLCReducer is analyzed. To analyze the impact algorithm is applied on five different java based projects and the amount of cohesion before and after applying the algorithm is shown in the paper. Also the results are compared with an existing approach.***

Keywords: Cohesion, coupling, PLCoh, PLCReducer

I. INTRODUCTION

Coupling and cohesion are among the basic metrics to measure the quality of software design. Coupling is the measure of the strength of association established by a connection from one module to another [1]". Therefore, the difficulty to understand the design and maintain the software increases as the coupling increases. In reverse of the coupling, high degree of cohesion results in high degree of understandability of the modules of a project[2]. A number of software quality attributes like maintainability, understandability, reusability [3, 4, and 5] are directly or indirectly depends upon the degree of coupling and cohesion. Reusability on one of the desirable feature by the developers and it can be achieved highly by minimum level of coupling. [6, 7, and 8]. 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 [9]. Any compromise with the cohesion strength impacts the quality of the software.

In this paper the impacts of 'PLCReducer' are analyzed. PLCReducer 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. But only reduction is coupling cannot insures the improvement in design complexity.

In the following sub-section there is a brief discussion about PLC(a coupling metric) PLCoh(a cohesion metric) that is used to understand the impacts of PLCReducer on cohesion.

A. Brief about PLC and PLCoh

To observe the impact of PLCReducer, a package level coupling metric PLC [10] and cohesion metric PLCoh [12] 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.

A1. PLC Formulation

It is assumed that there are N number of packages in the software. If P is the set of packages then P= {P1, P2, P3, Pi.....PN}. Then, PLC for the package Pi, is expressed as shown in equation 1.

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

$$Cohesion\ of\ Class\ C_i\ (C_iCoh) = \frac{Number\ of\ Relationships\ (directly\ and\ /\ or\ Transitively\ Related\ With\ C_i)}{(N-1)} \quad \text{..... ..... (2)}$$

Self relationship is not considered here thus in the denominator value in equation 2 is considered as (N -1) where N is the number of classes in the package. PLCoh of package P is normalized value of C_iCoh for i = 1 to N. Mathematically PLCoh is expressed as follows:

$$Package\ Lavel\ Cohesion\ (PLCoh) = \frac{\sum_{i=1}^{N} C_iCoh}{N} \quad \text{..... ..... (3)}$$

A2. PLCoh Formulation

Cohesion is the degree of relationship in a component. A package consists of classes. Thus for any class C_i that belongs package P having R_i be the number of relationships that are either directly or transitively related with this class C_i and N be the total number of classes in package P then, Cohesion of Class C_i is expressed as shown in equation.

Revised Manuscript Received on June 05, 2019

Aprna Tripathi, Department of Computer Engineering and Applications, GLA University Mathura (Uttar Pradesh), India.
Varsha Kumari, Department of Computer Engineering and Applications, GLA University Mathura (Uttar Pradesh), India.
Nikhil Govil, Department of Computer Engineering and Applications, GLA University Mathura (Uttar Pradesh), India.

The paper is divided into five sections. Section 2 presents the state of art related with coupling reduction. In section 3 the impact on PLCoh after applying the suggestions is analyzed. A comparative study is discussed in section 4. The proposed work is concluded in conclusion section.

1. Sate of Art

This section discusses the major work done in the area of coupling reduction approaches and its impact on the projects cohesion.

High degree of coupling is undesirable while zero coupling is also not ood for software design [11]. Thus, only here is one possibility to control the coupling. Many researchers suggested the approaches to control the coupling. The most popular way to control the coupling is to avoid 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.
Lung et al. proposes a approach based on clustering to restructure a low cohesive function into high cohesive function but there is no discussion about impact on coupling [14, 15]. A. Alkhalid et al. [16] proposed a method to reduce the software complexity for which they uses adaptive k-nearest neighbour (A-KNN) clustering algorithm by refactoring the packages based on similarity factor. Based on the similarity values they suggested moving a class from one package to other package. 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 [13]. Any compromise with the cohesion strength impacts the quality of the software if only focusing on the coupling of the entire system.

2. Impact of PLCReducer on PLCoh After Employing Suggestions

As proposed in the previous section, when suggestions recommended by PLCReducer are implemented in the existing system, methods may get shifted from class of one package to class of other packages. In this process, it is also possible that whole class of one package is shifted to other package. It should ensure that coupling reduction should not lead to cohesion reduction.

To verify that PLCReducer only reduces the coupling of the system and not to the cohesion, we apply the algorithm PLCReducer on four java based projects and analyze the PLCoh and Modified PLCoh (MPLCoh) after employing the suggestions. Table 1 show that after employing the suggestions of PLCReducer, either PLCoh increases or remain unchanged. This can be also visualized in the figure 1.

Table 1. PLCoh and MPLCoh before and After Employing Suggestions

Table 2.

Project	LOC	Package	PLCoh	MPLCoh	Project Average		% increase/decrease in PLCoh
					PLCoh	MPLCoh	
Javaopration	899	TestCase	.401	0.401	0.15	0.16	6.67
		Parser	0.05	0.078			
		SourceCode	0.0	0.0			
Lamistra	4932	Editor	0.25	0.5	0.416	0.467	12.2
		Stratgo	0.30	0.3			
		Server	.875	.875			
		Player	.659	.659			
		Remote	0.0	0			
AnagramGame	395	Lib	0.0	0	.25	0.25	0
		Ui	0.5	0.5			
AnagramGame _Modified	402	Lib	0.0	0.0	0.25	0.25	0
		Ui	0.5	0.5			

Shipment	792	Shipment	0.04	0.04	0.02	0.146	63.0
		ShipmentCompany Detail	0	0.25			

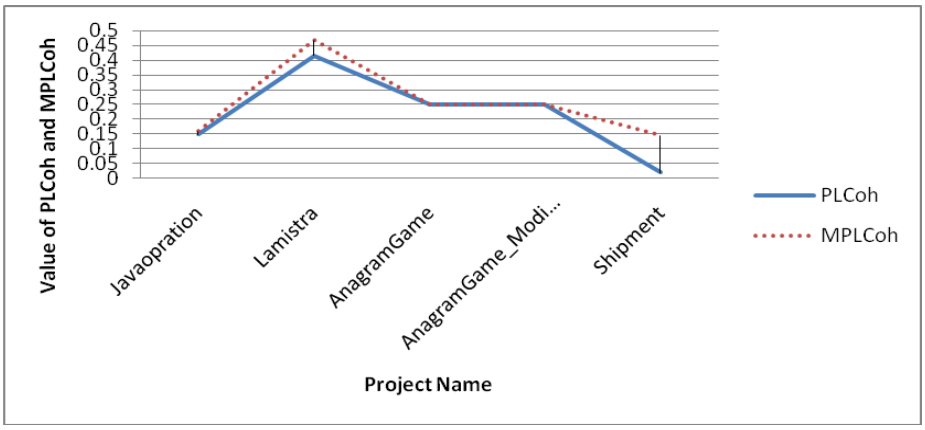

Figure 1. Average PLCoh and MPLCoh of Projects

3. Comparative study

To establish the validity of the proposed algorithm PLCReducer, results obtained from project shipment for PLCReducer are compared with the approach proposed by Alkhalid et al. The detailed structure of project shipment is discussed in Table 2.

Table 3. Shipment packages and the classes inside the each package

Package ID	Package	Class ID	Class
P1	Shipment	C1	Demo
		C2	Segwhite
		C3	Segfor
		C4	shipment
		C5	ContactDetails
		C6	AboutCompany
		C7	Dependency
		C8	DomParser
		C9	CompanyBranch
		C10	seg
P2	ShipmentCompanyDetail	C11	Enquiry
		C12	colorBox
		C13	Box
		C14	BoxWeight
		C15	Feature

4.1. Implementation of Alkhalid et al. Approach

To find the results for shipment using Alkhalid et al., we computed the values as discussed in [16].
The similarity between a class and package is computed using the equation 4 and a similarity metric is generated.

$$Sim(A,P) = w \quad \text{..... (4)}$$

Where A is the class; P is the package; w is the number of instances of class A used as attributes by the classes in the package P. For this we first calculated the number of instances of each class inside all other classes as shown in figure 2.

Class/ Package		C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15
		P1	P1	P1	P1	P1	P1	P1	P1	P1	P1	P2	P2	P2	P2	P2
C1	P1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C2	P1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C3	P1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C4	P1	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0
C5	P1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
C6	P1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C7	P1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C8	P1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C9	P1	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0
C10	P1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C11	P2	1	0	0	0	1	1	0	0	1	0	0	0	0	0	0
C12	P2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
C13	P2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C14	P2	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0
C15	P2	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0

Figure 2. Number of instances of each class in all the
Figure 3. other classes

Table 3 shows the number of connections between the classes inside each package and the number of connections to other packages.

Table 4. Number of connections in the original source code of shipment project

	No. of connections inside the package	**No. of connections to other package**
Shipment	4	3
ShipmentCompanyDetail	7	4
Total	11	7

Similarity matrix consists of two columns representing two packages and 15 lines representing 15 classes in those packages as shown in Table 4.

Table 5. Similarity matrix between each class and all other packages

Class	**Package Name**	**Shipment**	**ShipmentCompanyDetail**
	Package ID		
C1	P1	0	0
C2	P1	0	0
C3	P1	0	0
C4	P1	0	3
C5	P1	1	0
C6	P1	0	0
C7	P1	0	0
C8	P1	0	0
C9	P1	3	0
C10	P1	0	0
C11	P2	4	0
C12	P2	0	1
C13	P2	0	0
C14	P2	0	2
C15	P2	0	4

Based on the values of similarity metrics and alkhalid et al. approach changes are proposed that are summarized in table 5.

Table 6. Suggested changes by Alkhalid et al. Approach

Class	Old package	New package
Enquiry	shipmentCompanyDetail	Shipment
shipment	Shipment	shipmentCompanyDetail

After implementing the changes as per the table 6 the number of connection inside and from other projects are summerized in table 6.

Table 7. Changes in connections number after applying Alkhalid et al. approach

Package Name	No. of connections inside the package	No. of connections to other package
Shipment	0	0
ShipmentCompanyDetail	+3	4
Total	+3	-3

4.2. Results comparision

Table 7 summarizes the results computed by alkhalid et al. and PLCReducer as shown in table 1. Results shows that the suggetions derived from PLCReducer are more capable to reduce the coupling as well as increase the cohesion.

Table 8. Comparative result analysis of Impact on coupling and cohesion

	% decreace in coupling	%increase in cohesion
Alkhalid Approach	42	27
PLC Ruducer	65	63

II. CONCLUSIONS

This paper 'PLCReducer' that is used to reduce the PLC in case of faulty design is analyzed with the reference of cohesion. It is found that either PLCoh increases or remains unchanged, thus validating the 'PLCReducer' algorithm. Also the results are compared with an existing approach which shows that the suggestions generated by PLCReducer are more capable to increase cohesion and reduce coupling. Proposed approach is also compared with an existing approach [16] and it is found that the suggestions generated by PLCReducer are more capable to reduce the design complexity.

REFERENCES

1. W. Stevens, G. Myers, L. Constantine, Structured Design, IBM Balagurusamy, Programming in ANSI C, Tata McGraw-Hill Education, 2008, ISBN 9780070648227
2. Systems Journal, 13 (2), pp. 115-139,1974.
3. 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.
4. 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.
5. 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
6. Gui Gui and Paul D. Scott. "Ranking reusability of software components using coupling metrics", *Journal System and Software*, pp. 1450-1459, 2007.
7. 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.
8. 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
9. M. Fowler. "Reducing coupling", IEEE software, 2001.
10. Tripathi, A. & Kushwaha, D.S. “A metric for package level coupling” CSIT (2015) 2: pp 217–233
11. 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
12. Martin R., "Object Oriented design quality metrics: an analysis of dependencies", ROAD, 1995.
13. Hautus E., "Improving Java software through package structure analysis", In Proc. International Conference on Software Engineering and Applications, Cambridge, USA, pp. 4 - 6, 2002.
14. Lung, C.-H., Xu, X., Zaman, M., Srinivasan, A.: ‘Program restructuring using clustering techniques’, J. Syst. Softw., 2006, 79, (9), pp. 1261–1279
15. Lung, C.-H., Zaman, M., Nandi, A.: ‘Applications of clustering techniques to software partitioning, recovery and restructuring’, J. Syst. Softw., 2004, 73, (2), pp. 227–244
16. A. Alkhalid, M. Alshayeb and S. A. Mahmoud, "Software refactoring at the package level using clustering techniques," in IET Software, vol. 5, no. 3, pp. 276-284, June 2011.