IJASRM
International Journal of Advanced Scientific Research and Management, Volume 3 Issue 5, May 2018
www.ijasrm.com
ISSN 2455-6378

Improving Object Oriented Software Quality Using Inheritance Coupling Complexity Metric At Package Level(ICmP)

Vanitha N[1], Thirumalaiselvi R[2]

[1] Research Scholar, Research and Development Centre, Bharathiyar University, Coimbatore, Tamilnadu, India and
Assistant Professor, Department of Computer Science, Women's Christian College, Chennai, Tamilnadu, India

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

Abstract

Determining the complexity of the software project is not only just enough without identifying the part of the program causing serious damages to the software. Identifying the highly central class is essential as it becomes a single point of software failure and can be fixed. As any modification in the base class will affect the derived class, the determinant classes must be identified and changes should be done without affecting the centralized class causing any serious damaged to the software. We compare the actual and expected complexity values of object oriented projects determined based upon the inheritance level in package hierarchy and improve the results by reducing the complexity of projects by detecting the centralized classes that cause software defects due to coupling and redesign the code without affecting the centralized class in the earlier phase of software developmental life cycle.

Keywords: *Complexity, Determinant Classes, Modification Packages, Modification*

1. Introduction

In the last ten decades of Software Engineering, Object –orientation developed into the most popular programming and design approach. Software development techniques and methods have to cope with the growing complexity and size of software applications. Software developers engage themselves to develop better software in time. Software quality is achieved when the software is free of defect. The evaluation of complexity in software system is significant as it associated with software defects and it remains a challenge for software developers, engineers and testers. Coupling is one of the most significant aspects of code which impacts software quality. Defects are more likely to appear in the portion of the program that expose high coupling and can increase the overall complexity of the software. Determining the complexity of the software project is not only just enough and the part of the program causing serious damage to the software should be identified and fixed. These should be done during the development phase of the software paradigm. Otherwise, the testers focus on solving the impact that may arise due to modification of the program. To avoid this situation, the overall design of a software system can be evaluated before the modification.

The main focus of the present research work is to compare the software complexity determined using ICmP metric proposed by [15] and improve the software quality by reducing the complexity of the software programs. It is based upon the centralized class affecting the software and redesign the code in the part that cause damages to the software without affecting the centralized classes and thus put forth some effort to increase the overall quality of the software artifact.

IJASRM
International Journal of Advanced Scientific Research and Management, Volume 3 Issue 5, May 2018
www.ijasrm.com
ISSN 2455-6378

The remainder of this paper is organized as follows. *Section II* provides the research methodology *Section III provides* a background to the field and presents some relevant surveys. *Section IV* provides data collection. Analysis and results of the study are presented in *Section V.Section VI* concludes the paper.

2. Methodology

The study follows the Braind et al. [2] framework and systematic approach to study on the dependency analysis and CK metrics follows the approach identified by Chidamber and Kemerer[3] .

2.1 Hypothesis

The goal of this study is to compare the actual and expected complexity values of object oriented projects determined based upon the inheritance level in package hierarchy and improve the results by reducing the complexity of projects by detecting the centralized classes that cause software defects due to coupling and redesign the code without affecting the centralized class in the earlier phase. This analysis is based on the hypothesis given in TABLE 1.

Table 1. Hypothesis Addressed

	Hypothesis
H1	Centralized defect prone classes associated with the software quality

2.2 Inclusion Criteria

A study of Journal paper or Conference proceedings published in English to be included in this review. From the publication of similar studies, only the most related or recent to be included.

2.3 Identification of Papers

Included papers were published between the year 2010 and 2018. Key word search, using the search engines were Google Scholar, Scopus, IEEExplore and ScienceDirect. These search engines covered the majority of software engineering publications and the search string used for this is given in the references. Totally 25 papers were identified, 8 papers were rejected as not relevant to this research and included 17 papers finally.

3. Background and Related Work

This paper compares and reduces the complexity using the proposed ICmP metric by the authors Vanitha et al. [15]

This paper uses the object-oriented metrics presented by Chidamber et al. [3] to identify and analyze the coupled components.

This paper identifies the centralized defect prone classes presented by Vanitha et al. [14]

Subramanyam and Krishnan [11] presented a survey on eight more empirical studies, all showing that OO metrics are significantly associated with defects.

Bavato et al. [1] discussed about improving modularization.

4. Data Collection

As coupling is used to determine the determinant and dependent classes, the necessary data have been collected from small scale Java projects. The detailed study is done on the Examination External Remuneration System and to compare and interpret the result, two more open source Java projects have been used, Airline Reservation System and Google Mapping System. Table 2 shows the number of packages and classes in three different projects.

Table 2. Number of Packages and Classes

Projects	Examination External Remuneration System	Airline Reservation System	Google Mapping System
No. of Parent packages	2	3	2
No. of Sub Packages	6	7	8
No. of Classes	31	58	62

For detailed study Table 3 shows the packages and sub packages in bold and its respective classes of the Examination External Remuneration System Project.

Table 3. Packages, Sub Packages and its Classes of Java Project

Packages	Exam	Exam.Examiner_Details.java Exam.Student_Details.java Exam.Mark_Details.java Exam.Result_Details.java
	Remun	Remun.Valuator_Details.java Remun.Valuation_Details.java Remun.Calc_Details.java
	Adm	Exam.Adm.Control.java Exam.Adm.Secur.java Exam.Adm.Net.java Exam.Adm.Del.java Exam.Adm.Update.java
	Log	Exam.Log.Get.java

Sub Packages		Exam.Log.Set.java Exam.Log.User.java Exam.Log.Det.java Exam.Log.View.java Exam.Log.Check.java Exam.Log.Sub.java
	External	Remun.External.Detai.java Remun.External.Amt.java Remun.External.Eval.java Remun.External.Loc.java
	Internal	Remun.Internal.Count.java Remun.Internal.Paym.java
	Details	Remun.Details.Staff.java Remun.Details.Subj.java Remun.Details.Dat.java
	Payment	Remun.Payment.Prev.java Remun.Payment.Curr.java Remun.Payment.Info.java

5. Analysis and Experimental Results

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. According to the authors previous work Vanitha et al. [15] weights are assigned based on the level of inheritance at package level. 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 4. 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 IV.

Table 4. Assignment of Weights

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

Table 5 shows the coupled classes in different package level of the Project, Examination System Software along with their weights.

Table 5. Coupled Classes of the Project with their Respective Weights

Base Class	Derived Class	Weight
Examiner_details	Student _details	1
Valuator_details	Calc_details	1
Net	Student _details	2
View	Result_details	2
Net	Del	3
Set	User	3
View	Get	3
User	Get	3
Examiner_details	Count	4
Examiner_details	Subj	4
Mark_details	Prev	4
Valuation_details	Get	4
Valuation_details	Net	4
Valuator_details	Det	4
Valuator_details	Sub	4
Secur	Amt	5
Secur	Subj	5
Prev	User	5
Subj	User	5
Info	Del	5
Update	Subj	5
Set	Count	5

As any modification in the base class will affect the derived class, the base class is considered as the determinant class and derived class is the dependent class. The determinant class can be a highly centralized defect-prone component. According to our tactics, the highly centralized defect-prone class in the software system has been found out based on the number of times the class gets inherited and how deeper its level in a package hierarchy. In the same way, the most dependent class in the software system has been found out based on the number of times it inherits and how deeper its level in a package hierarchy. The sample experimental results for identifying the centralized defect-prone class and dependent class are as follows.

Table 6 shows the dependency value of dependent classes whose occurrences are more than once. Dependency value is obtained by adding the weights assigned to the dependent classes based upon their position in the package hierarchy for the number of occurrences.

Table 6. Dependency Values of the Dependent Classes

Dependent Class	No. of Occurrence	Dependency Value
Student_details	2	3
Del	2	8
Get	3	10
User	3	13
Count	2	9
Subj	3	14

It is shown from the Table 6, that for the project, Subj is the dependent class which has the highest

dependency value of 14. Hence it is identified that the most dependent component of the Project is Subj. The finding of the most dependent component of the project is essential as it is the one which highly gets affected whenever any change happens in the determinant classes.

Identifying the highly central component is essential as it becomes a single point of software failure and thus serious damages to the software can be fixed. So, as similar for dependent classes, the dependency values of the determinant classes are also obtained.

It is shown from the Table 7, that the highest dependency value of the base class is 10. Hence it is identified that the defect-prone centralized component of the Project is 'Secur'. The identification of these classes eases modification which may reduce the software defects.

Table 7. Dependency Values of Coupled Classes

Determinant Class	No. of Occurrence	Dependency Value
Examiner_details	3	9
Valuator_details	3	9
Valuation_details	2	8
Secur	2	10
Net	2	5
Set	2	8

H1: Centralized defect prone classes associated with the software quality

Once we identified the centralized defect prone class, the complexity of the object oriented system that happened due to the inheritance coupling at package level can be reduced without affecting the centralized class as the any changes done in the highly determinant class damages the overall software quality.

From the research work presented by vanitha et al. [14] the same Java project, Examination External Remuneration System is chosen to demonstrate how the changes to be made without affecting the other part of the program in detail.

According to the researcher, it has been found that the defect-prone centralized class of the Examination system Project is 'Secur'. By assessing the position of the class 'Secur' in the package hierarchy, the total weight assigned for the class 'Secur' is 10 is identified.

To reduce the complexity level, analyze the classes which depend on the class 'Secur'. The classes Amt and Subj are highly dependent on it. Among these two classes, the class 'Subj' is inherited by another class 'User'. Therefore, no modification should be made on the part of the program where the class 'Subj' inherits from the class 'Secur' . The removal of the class 'Amt' does not affect the other part of the program since it is not being inherited by any other class. Similarly the class 'Get' in the different subpackage extends the class 'Valuation_Details" in a different package and not inherited by any other classes it has been redesigned in order to lower the complexity of the project.

The weight has been reduced after redesigning, the classes in the given projects are shown in Table 8.

Coupled Classes at Package level	Examination External Remuneration System	Airline Reservation System	Google Mapping System
Classes extends from the same package	2	3	3
Classes in sub package extends from the same package	2	5	4
Classes in sub package extends classes of sub package belongs to same package	3	2	2
Classes in package extends classes of sub package belongs to other package	5	9	6
Classes in sub package extends classes of sub package belongs to other package	5	5	9

Table 8. Number of Classes at Different Package Level After Modification

It has been proved that the complexity of software projects can be reduced by determining the most determinant centralized error prone classes and modifying the project without affecting the other classes which depend on the most determinant classes.

Table 9 shows the complexity of three projects before and after modification.

Table 9. Complexity Values Before and After Modification

Figure 1. shows the actual complexity value before modification , its expected value and value after modification for three java projects.

Projects	Actual Value before modification	Expected Value	Value after modification
Examination External Remuneration System	81	51	60
Airline Reservation System	113	72	80
Google Mapping System	104	72	86

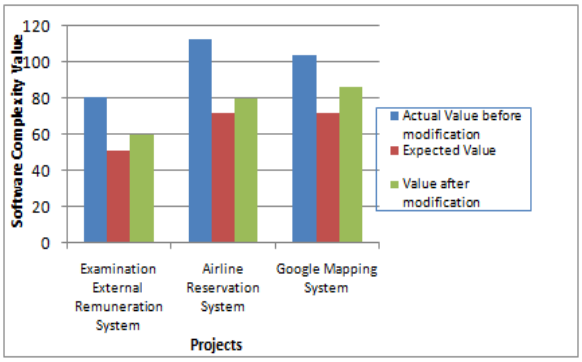

Fig 1. Actual Complexity value before and after modifications

To evaluate the improvements happened in the project, we used the improvement percentage statistical formula which helps in evaluating the performance growth. Table 10 shows the mean improvement percentage of three different projects before and after modification is 25.3.

Table 10. Improvement Percentage from Actual Complexity Value to Modified Complexity Value

Projects	Actual Value before modification	Value after modification	Improvement Percentage
Examination External Remuneration System	81	60	25.9
Airline Reservation System	113	80	29.2
Google Mapping System	104	86	20.9
		Mean	25.3

From the result ,it has been proved that the ICmP complexity metric helps in determining the error prone classes and based on it, the changes can be made in the software projects and shows the improvements thus happened.

6.Conclusion and Future Work

It is concluded that determining the centralized error prone classes will help to reduce the complexity of the software product. The result helps the software developers and researchers to develop quality software.
In future, progress of this study can be done on large scale object oriented projects.

References

[1] Bavota G, De Lucia, A, Marcus, A, Oliveto, R, "Using structural and semantic measures to improve software modularization", Emprical Software Engineering, 18(5), 901-932, (2013).
[2] Briand, Lionel, Sandro Morasca, and Victor R. Basili. "Defining and validating high-level design metrics." ,(1994).
[3] Chidamber S, Kemerer , C. A metrics suite for object-oriented design [J]. IEEE Transactions on SoftwarenEngineering, **20**(6): 476-493,(1994).
[4] Gupta, V, Object – oriented Static and Dynamic Software Metrics for Design and Complexity,(2010).
[5] Huang P, Zhu J. Predicting the fault-proneness of class hierarchy in object oriented software using a layered kernel [J].Journal of Zhejiang University - Science A, **9**(10):1390-1397,(2008).
[6] Jitender Kumar Chhabra and Varun Gupta, "A Survey of Dynamic Software Metrics", Journal Of Computer Science and Technology, 25(5), 1016-1029 ,(2010).
[7] Karine Mordal, Nicolas Anquetil, Jannik Laval, "Alexander Serebrenik,Bogdan Vasilescu, Stephane Ducasse" , Software Quality Metrics Aggregation In Industry, Journal Of Software: Evolution And Process,25(10), 1117-1135, (2013).
[8] Mallikarjuna, C, Sudheer Babu, K, Chitti Babu, P, "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).
[9] Nagappan, N and Ball,T, "Using Software Dependencies and Churn Metrics to Predict Field Failures: An Empirical Case Study,"in

International Symposium on Empirical Software Engineering and Measurement, pp.364-373,(2007).

[10] Rohitt Sharma, Paramjit Singh, Sumit Sharma, "An Approach Oriented Towards Enhancing a Metric Performance", International Journal On Computer Science And Engineering (IJCSE), 4(5), 743-748, (2012).

[11] Subramanyam R and M.S. Krishnan, "Empirical analysis of ck metrics fir object-oriented design complexity: Implications for software defects" IEEE Trans. Software Eng.,vol.29,pp.297-310, (2003).

[12] Tahir, MacDonell, S.G., "A Systematic mapping study on dynamic software quality metrics", Proc. 28th IEEE International Conference on Software Maintenance ,Riva del Garda, Italy, 326-335,(2012).

[13] Tolga Pusatli,O, Sanjay Misra , "A Discussion On Assuring Software Quality In Small And Medium Software Enterprises: An Empirical Investigation", Portal of scientific journals of croatia,18(3),447-452,(2011).

[14] Vanitha N and ThirumalaiSelvi R , "Identification and Visualization of Determinant and Dependent Classes Based on its Inheritance Level in Package Hierarchy", Journal of Advanced Research in Dynamical and Control Systems, 02–Special Issue, 732-740, ISSN: 1943-023X ,(2018).

[15] Vanitha, N., & Thirumalaiselvi, R., Inheritance Coupling Complexity Metric in Association with Modifiability at Package Level: An Empirical Exploration. International Journal of Pure and Applied Mathematics, 118 (18), 3789-3797, (2018).

[16] Zimmerman, Thomas, Nachiappan Nagappan, Kim Herzig, Rahul Premraj, and Laurie Williams. "An empirical study on the relation between dependency neighborhoods and failures." In Software Testing, Verification and Validation (ICST), 2011 IEEE Fourth International Conference on, pp. 347-356. IEEE, (2011).