Software Analysis with Package Level Modularization Metrics

[1]MOHSIN KHAN SHAIKH*, [1]MUHAMMAD ADIL ANSARI,
[2]AKHTAR HUSSAIN JALBANI

[1]*Quaid-e-Awam University College of Engineering, Science & Technology, Larkana*
[2]*Quaid-e-Awam University of Engineering, Science & Technology, Nawabshah*

**Corresponding Author's email: engr_mohsin@quest.edu.pk*

Abstract

Increasing complexity of software systems require extensive maintenance through decomposition of source code into appropriate abstractions to achieve effective modularization. An optimal modularization of object-oriented software insuring low coupling and high cohesion is perceived as a challenging task. In this paper, we present empirical perspective of new inter-module coupling based modularization metrics to assess their utility. In particular, we explore impact of correlation between these design-based modularization metrics and external quality attributes of software systems. Our experimental study covers 34 open source java software systems and shows that inter-module coupling based modularization correlates with existing metrics of modularization and also bear substantial relationship with vital quality attributes of software design.

Keywords: Software metrics, Software analysis, Modularity

INTRODUCTION

Software evolution process is carried out through continuous addition, modification and re-organization of source code. As these activities are reflected into source code, there is an eventual drift in software design. Recently, there has been significant advancement to reverse engineer the software systems for automatic extraction of its design depicting an aggregate view. Some of notable techniques in this regard are related to partitioning of software systems into sub-systems (clusters) [1]. With an advancement in building tools and methodologies for software maintenance, there has been significant research over mechanism of partitioning the software into sub systems considering source code abstractions like classes and packages [2]. In particular, package organization provides higher abstraction and easier way for comprehension, complexity reduction and understanding maintainability. We also build linear correlation model with existing validated modularity metrics studied in different domains of engineering [3]. We evaluated this approach with comprehensive empirical analysis over open source java systems. Consequently, these findings help to evaluate the cohesive quality of software. Additionally, significant statistical relationship was also witnessed with

maintainability, design deficit and testability. Thus, this effort can help taking measures to minimize the design flaws of software systems. However, due to frequent changes into software, increase in fragility of software modularization is not an impossible occurrence.

Sarkar *et al*. have proposed a new modularization metrics suite based on packages as its functional component [4]. In this context they have devised two categories of coupling based metrics, i.e., based on inheritance or association and method invocation. As a matter of fundamental perspective of modularization, inheritance and association produce abstraction and encapsulation in software design.

In this paper, we explore inheritance and association-based coupling metrics for automated optimization of module structure and determine their impact over testing efforts, deficit produced in design of software and its overall emphasis.

METRICS STUDIED

This section provides the description and summary of investigated metrics and software quality attributes. Table 1, 2, 3 summarizes the definition of inter-module coupling modularization metrics produced with specific methodology of programming design, i.e., inheritance, association and polymorphism. With object-oriented design paradigm, inheritance and association are the most important dependence relationship showing interactive coding structure among packages and classes. In programming sense, inheritance relationship is created when a class extends another class, while association is formed when class uses another class. In addition to this, such dependencies among the classes are often seen to be distributed in different modules (Packages in our context of study). Another coupling relationship known as fragile base-class problem follows a design phenomenon when two classes existing different module show interdependencies causing fragility into base class. Sarkar *et al*. describe these metrics to measure the modularization quality of modules showing the strength of package organization in software structure. Furthermore, illustration of each metric is given as under:

- ***IC(S)***: is a composite metric that measures the extent to which inheritance-based dependencies among and within the packages are minimized.
- ***AC(S)***: is composite metric that measures the extent to which association-based dependencies among and within the packages are minimized.
- ***BCFI(S)***: is a composite metric that measure the extent to which polymorphic design of methods is restricted to the defining packages.

Table 1: Coupling Metrics

Metric	Definition
Inheritance based Inter-Module Coupling	$IC(S) = \frac{1}{\|p\|} \sum IC\ (P)$
Association Induced Intermodule - Coupling	$AC(S) = \frac{1}{\|p\|} \sum AC\ (P)$
Base-class Fragility	$BCFI(S) = \frac{1}{\|p\|} \sum BCFI\ (P)$

Table 2: Modularity Metrics

Metric	Definition
M_{newman}	Modularity and community structure in network [5]
MQ	Modularity of software based on clustering [6]
$M_{g\&g}$	Modularity of mechanical products. [7]
M_{rcc}	Modularity based on dependency cost [8]

Description of quality metrics is as under:

- ***MI***: Maintainability index is composite metric that incorporates number of traditional source code metrics into single value that indicates relative maintainability.
- ***QDI:*** Quality Deficit Index is a positive value aggregating the detected design flaws (i.e., code smells and architectural smells).
- ***TLOC:*** Testability metric in our study is considered as effort required to test the software system.

Table 3: Quality Metrics

Metric	Definition
Halstead Maintainability Index (MI) [9]	$MI = 171 - 5.2 \times \log(aveV) - 0.23 \times aveV(g') - 16.2 \times \log(aveLOC)$
Quality Deficit Index (QDI) [10]	$QDI = \frac{\sum_{all-flaw-instances} FIS_{flaw-instances}}{KLOC}$
Testability Metric (TLOC) [11]	$TLOC = \sum_{i=1}^{n} LOC(Ci)$

EXPERIMENTAL STUDY

In this section, we describe our comprehensive experimental evaluation of Sarkar's et al. modularization metrics over open source software system. The key process is selection of intelligent metrics to evaluate software modularity. Therefore, most efficient and already utilized methodology in our research. Our objective is to analyze the effects of coupling modularization metrics over different quality attributes of software systems. i.e., modularization metrics, external quality attributes. Following study features are listed as objectives of experimental work.

1. Assess strength of relationship between Sarkar's coupling modularization metrics and Baseline modularization studied in different domains.

2. Determine impact of Sarkar's coupling modularization metrics over maintenance, design flaws and testing effort.

Develop the theoretical perspective for Sarkar's coupling modularization metrics in improving overall software quality. We selected 34 versions of three different open source system in our experiment. JHotDraw [12]: a Java GUI framework for technical and structured Graphics. Ant [13]: a Java library and command-line tool whose mission is to drive processes described in build files. Google-Web Toolkit [14]: an open source set of tools that allows web developers to create and maintain complex JavaScript front-end applications in Java.

These systems have reasonable size, manageable degree of complexity, diverse application domain and easily accessible source code for data processing and computation of described metrics in order to carry out further study. To calculate metrics, coupling modularization metrics, TLOC and MI, source code of subject system was parsed and analyzed through commercial tool Understand: A commercial static analysis tool. To figure out QDI, we used evaluation version of open source tool Infusion. After collecting metrics data, statistical test of correlation is used to obtain the objectives defined. The subject of software sustainability is emerging as bench-mark to realize applications of software in social, economic, operational and technical terms. Hence, relevant empirical studies are required to explore the subject further. We presented an experimental analysis over 34 versions of three different open source java systems that includes research objectives, design, data processing and experimental results.

EXPERIMENTAL RESULTS

In Table 4 and 5, magnitude of association of coupling based metrics is shown at different strength of significant levels with bold values into table cells. Modularity metrics correlations are shown in Table 4.

Table 4: Correlation with Quality Metrics

Project	Metric	MI	QDI	TLOC
JHotDraw [9]	BCF(S)	-0.86	0.92	0.74
	IC(S)	-1.000	0.99	0.78
	AC(S)	-0.18	0.32	0:14
Apache-ant [14]	BCF(S)	-0.072	0.75	0.60
	IC(S)	-0.52	-0.37	-0.93
	AC(S)	-0.51	0:52	0:39
Google-Web Toolkit [11]	BCF(S)	-0.19	0:49	0:085
	IC(S)	0.45	0.024	-0.80
	AC(S)	-0.58	0:01	-0.87

Table 5: Correlation with Modularization Metrics

Project	Metric	M_{newm}	M_{bunch}	$M_{g\&g}$	M_{rcc}
JHotDraw [9]	BCF(S)	**0.94**	0.77	0.59	**0.97**
	IC(S)	**0.94**	0.66	0.24	0.84
	AC(S)	0.53	0.72	0.78	0.70
Apache-ant [14]	BCF(S)	0.41	0.20	0.24	0.60
	IC(S)	0.27	0.76	-0.79	-0.93
	AC(S)	0.13	-0.54	0:54	0:31
Google-Web Toolkit [11]	BCF(S)	0.26	0.32	0.12	-0.45
	IC(S)	**0.83**	**0.80**	0.45	-0.90
	AC(S)	-0.94	-0.90	0:41	**0.98**

Some important observations dealing with modularity correlation of Table 4 are described. For JHotDraw, BCF(S) is seen strongly correlated with M_{newm} and M_{rcc}. AC(S) is observed to be in significant relationship with modularity metrics of M_{bunch}, $M_{g\&g}$ and M_{rcc}. For Apache-Ant, strong association of correlation is established only for IC(S) with M_{bunch}, $M_{g\&g}$ and M_{rcc}. Interestingly, M_{newm} has shown weak statistical significance with most of coupling metrics in all cases. While in case of Google-Web ToolKit, AC(S) and IC(S) have produced

confident statistical relationship with modularity metrics. Such kind of variations in modularity correlation can be result of particular design paradigm of software systems where use dependencies are minimum in *JHotDraw* and extend dependencies are maximum in *Apache-Ant.*

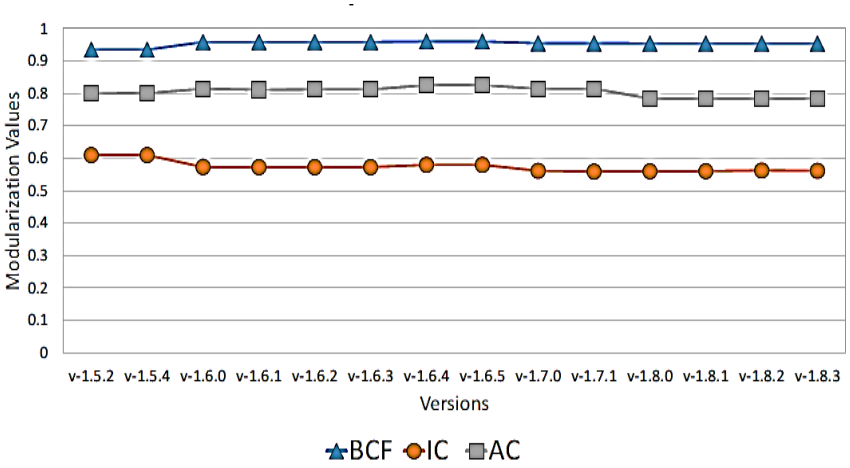

Figure 1: Apache Evolution Scenario

We can identify interesting observation of correlations from Table 5. First, BCF(S) is in strong and positive significant correlation with QDI and TLOC employing that fragility of base-class in software may result in the design deterioration and testing overhead. Second, in all cases, inheritance and association-based coupling metrics are inheritance and association-based coupling metrics are negatively correlated with MI.

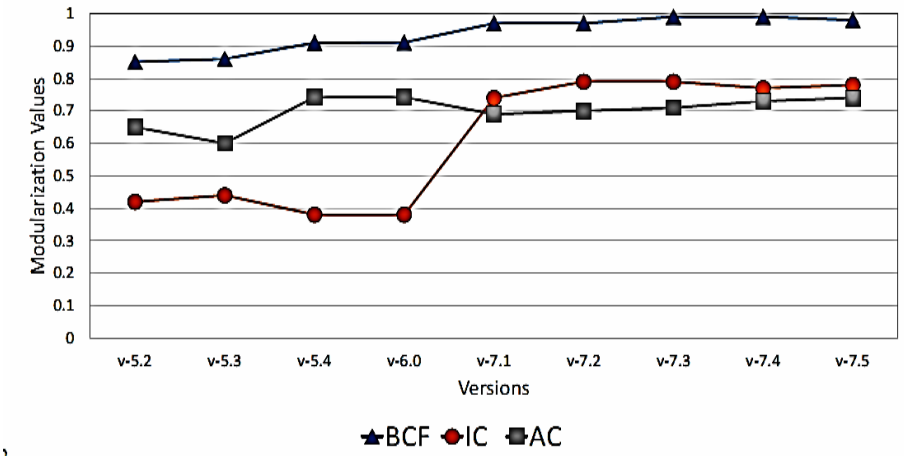

Figure 2: JhotDraw Evolution Scenario

In software engineering, process of evolution driven by incremental development and design changes. In this section, we present pattern of coupling metrics with consecutive

versions of software systems through graphical representation as shown in Figure 1, 2 and 3. In 14 versions of Apache-ant, all the coupling-based metrics are seen in consistent pattern. In 9 versions of JHot-Draw, AC(S) and IC(S) is observed to be improving. BCF(S) has improved in gradual manner to significant extent. In 11 versions of Google-Web Toolkit, BCF(S) and IC(S) has shown slight decline as the software evolves, while AC(S) has exhibited an improving trend.

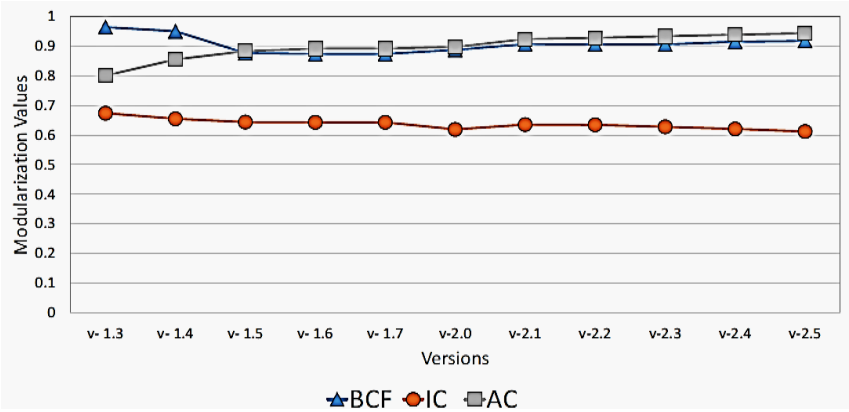

Figure 3: Google Webtool Kit Evolution Scenario

In this regard, we reported statistical relationship of Sarkar's et al. modularization metrics with existing validated modularity metrics studied in different domains of engineering and quality metrics. Indeed, from theoretical standpoint, our study has diverse quality assurance focus but with major emphasis on architectural sustainability. We do not rule out other parameters that may have arguably better explanatory power, however, our effort is to explore significance of package-based modularization metrics.

CONCLUSION

The modularity is an important aspect of software system describing its overall quality and strength. In this paper, we investigated package based proposed modularization metrics from multi-dimensional views using correlation methodology. Our empirical study is a statistical proof for utility and application of coupling modularity metrics through statistical correlation methodology. Our results show that coupling based modularization metrics have significant relationship with modularization metrics proposed in different engineering domains. Such findings can help the software engineer to develop design plan to ensure optimized source code architecture. Results over open source software system show that managing the coupling between the packages can also reduce maintenance work, deficit in design quality and testing effort.

REFERENCES

[1] B. S. Mitchell and S. Mancoridis, "On the automatic modularization of software systems using the Bunch tool," *IEEE Trans. Softw. Eng.*, vol. 32, no. 3, pp. 193–208, Mar. 2006.

[2] H. Abdeen, S. Ducasse, and H. Sahraoui, "Modularization metrics: assessing package organization in legacy large object-oriented software," in *18th Working Conf. Reverse Engineering*, 2011, pp. 394–398.

[3] S. Sarkar, A. C. Kak, and G. M. Rama, "Metrics for measuring the quality of modularization of large-scale object-oriented software," *IEEE Trans. Softw. Eng.*, vol. 34, no. 5, pp. 700–720, Sep. 2008.

[4] K. S. Lee and C. G. Lee, "Comparative analysis of modularity metrics for evaluating evolutionary software," *IEICE Trans. Inf. Syst.*, vol. 98, no. 2, pp. 439–443, 2015.

[5] M. E. J. Newman, "Modularity and community structure in networks," *Proc. Natl. Acad. Sci.*, vol. 103, no. 23, pp. 8577–8582, 2006.

[6] S. Mancoridis, B. S. Mitchell, Y. Chen, and E. R. Gansner, "Bunch: a clustering tool for the recovery and maintenance of software system structures," in *Proc. IEEE Int. Conf. Software Maintenance - 1999 (ICSM'99). "Software Maintenance for Business Change" (Cat. No.99CB36360)*, 1999, pp. 50–59.

[7] F. Guo and J. K. Gershenson, "A comparison of modular product design methods based on improvement and iteration," in *Volume 3a: 16th Int. Conf. Design Theory and Methodology*, 2004, pp. 261–269.

[8] A. MacCormack, J. Rusnak, and C. Y. Baldwin, "Exploring the structure of complex software designs: An empirical study of open source and proprietary code," *Manage. Sci.*, vol. 52, no. 7, pp. 1015–1030, 2006.

[9] K. D. Welker, "The software maintainability index revisited," *CrossTalk*, vol. 14, pp. 18–21, 2001.

[10] R. Marinescu, "Assessing technical debt by identifying design flaws in software systems," *IBM J. Res. Dev.*, vol. 56, no. 5, Sep. 2012.

[11] A. Tahir, S. G. MacDonell, and J. Buchan, "Understanding class-level testability through dynamic analysis," in *9th Int. Conf. Evaluation of Novel Approaches to Software Engineering (ENASE)*, 2014, pp. 1–10.

[12] Anonymous, "JHotDraw." [Online]. Available: http://www.jhotdraw.org.

[13] Anonymous, "Graphics. Ant." [Online]. Available: http://ant.apache.org.

[14] Anonymous, "GoogleWebKit." [Online]. Available: http://www.gwtproject.org.