Comparative Analysis of Modularity Metrics for Evaluating Evolutionary Software*

Ki-Seong LEE†, Nonmember and Chan-Gun LEE†, Member

SUMMARY Modularity is an effective evaluation approach for understanding the structural quality of evolutionary software. However, there are many diverse ways to measure it. In this paper, we analyze and compare various modularity metrics that have been studied in different domains to assess their applicability to evolutionary software analysis. Through extensive experiments with artificial DSMs and open-source software, we find that the correlations of those metrics are generally high despite their differences. However, our experiments show that a certain metric can be more sensitive to particular modular factors, hence applying of comprehensive modularity metrics must be taken into consideration.

key words: modularity, metric, software evolution

1. Introduction

Software evolution reflects changes in the functionality or complexity of software, which is formulated over time. Therefore, evolutionary software data are valuable resources that can guide the maintenance process. For effective software maintenance, we should be able to measure and compare software changes reliably. In order to evaluate the quality of evolutionary software, many studies have used the concept of modularity to understand inherent structural complexity. Modularity has been used in various areas such as social networks and biological networks, as well as software engineering. Many studies on modularity have proposed new metrics that quantify the degree of modularity [1]–[3]. Each study is similar in dealing with the structural relations of modules (also called clusters, groups or communities). However, there is no agreement among these studies on the criterion for modularity evaluation.

In this paper, we compare various modularity metrics studied in different domains, and analyze them empirically to apply the metrics to evolutionary software analysis. First, to examine the correlation of modularity metrics across different domains, we conduct experiments on various types of Dependency Structure Matrix (DSM) for which modularity changes from the best to the worst. We intentionally manipulate some structural factors of DSMs for the experiment and measure each case. Next, we perform similar experiments on open source projects to compare with the aforementioned artificial situations. Through this approach, we identify a series of factors influencing modularity and validate the feasibility of applying different modularity metrics to evolutionary software analysis. Section 2 shows modularity metrics and correlations on DSM. Section 3 presents the results of application to evolutionary software. Finally, Sect. 4 concludes our paper.

2. Comparison of Modularity Metrics

2.1 Modularity Metrics

The metric defined by Newman [1] is a well-known quantification approach of modularity in social network domain. We chose this particular metric because it has been recently applied to a large number of studies concerned with social network, metabolic network, neural network, the World Wide Web, etc. The key idea is that the number of edges (called links in [1]) within a module (called community in [1]) is greater than that of the expected ones and it is shown in Eq. (1). The number of edges between node $i$ and $j$ is represented by $A_{ij}$, which is either 0 or 1. $m$ is the number of total edges, and $k_i$ is edge degree of node $i$. $g_i$ represents the module containing node $i$. If $g_i$ and $g_j$ are the same module, $\delta(g_i, g_j)$ is set to 1, 0 otherwise. Since $k_i k_j / 2 m$ indicates the expected number of edges between node $i$ and $j$, this modularity captures the difference between the actual edges and the expected edges. We denote this modularity as $M_{newm}$.

$$M_{newm} = \frac{1}{2m} \sum_i \sum_j \left( A_{ij} - \frac{k_i k_j}{2m} \right) \delta(g_i, g_j)$$

Mancoridis et al. [2] proposed Modularity Quality (MQ) to evaluate the modularity of software systems. Originally, the MQ was defined as the sum of Cluster Factors (CF). However, in our study we use averaged CF [4] instead, which is normalized by the number of modules, to facilitate the comparison with other metrics. It is called $M_{branch}$ as shown in Eq. (2). $\mu_i$ represents the number of intra-edges of module $i$. The number of inter-edges between the modules $i$ and $j$ is $e_{ij}$. $k$ indicates the number of modules. Equation (2) states that the modularity is defined by considering both intra-edges and inter-edges.
\[ MQ = \sum_{i=1}^{k} \frac{2\mu_i}{2\mu_i + \sum_{j=1}^{k} (\epsilon_{ij} + \epsilon_{ji})}, \quad M_{bunch} = \frac{MQ}{k} \]  

Guo and Gershenson [3] proposed Eq. (3) in order to quantify the modularity of a mechanical product. Typically the structure of a mechanical product is less complex than that of a software system. Hence, we test whether the metric of [3] can be applied to software systems, too. In Eq. (3), \( M \) is the number of modules in the product and \( N \) is the number of total nodes (called components in [3]). \( R_{ij} \) represents dependency between \( i\)th row and \( j\)th column in a dependency matrix of the product. \( n_k \) is the index of the first node in module \( k \). Similarly \( m_k \) is the index of the last node in module \( k \). We denote this metric as \( M_{gkg} \) which is defined as the difference in edge density between intra and inter edges.

\[
M_{gkg} = \frac{M}{N} \left( \sum_{m_k \leq i < n_k} R_{ij} - \sum_{m_k \leq j < n_k} R_{ij} \right)
\]

Maccormack et al. proposed Dependency Cost in [5] and Milay et al. [6] normalized it to measure evolving software modularity, which is referred to as Relative Clustered Cost. It has value of 0 if all nodes have no dependencies, and value of 1 if each node is in a module of its own and all nodes are interdependent. We choose this metric because it has different perspective when it comes to optimal modularity. Since the metric depends on the degree of coupling, its high value represents high coupling which means low modularity in general. We therefore use its negated version, which is shown in Eq. (4), to compare with other modularity metrics in the same context. \( N \) is the number of total nodes, \( n \) is the size of the module containing both node \( i \) and \( j \), and \( \lambda \) is a user defined parameter which is set to 2 in [5]. \( d_{ij} \) represents an edge (dependency in [6]) between the nodes \( i \) and \( j \), and \( g_i \) is the module that includes the node \( i \). A node with many incoming edges is considered as a vertical bus. In a software system, typical vertical buses correspond to library classes which can be obstacles in modularizing the system. We have filtered top 10% nodes with highest in-degree into vertical buses as done in [6]. We shall denote the negated version of the Relative Clustered Cost as \( M_{rrc} \).

\[
\text{DependencyCost}(i, j) = \begin{cases} n^2d_{ij} & \text{if } \delta(g_i, g_j) = 1 \\ N^2d_{ij} & \text{if } \delta(g_i, g_j) \neq 1 \\ d_{ij} & \text{if } j \text{ is a vertical bus} \end{cases}
\]

\[
M_{rrc} = 1 - \frac{\sum_{i=1}^{N} \sum_{j=1}^{N} \text{DependencyCost}(i, j)}{N^2\lambda}
\]

2.2 Measuring Modularity on DSM

Chiriac et al. [7] argued that the modularity of a mechanical product is affected by the level of decomposition. For this, they generated modular matrices at various levels of granularity, and evaluated modularity for each case. Inspired by [7], we generated artificial modular DSMs for comparing and analyzing modularity metrics. At first, we created dependency graphs which consist of nodes and edges, and derived DSMs from the graphs. We then made various changes to the DSM. Major factors that we controlled included the size and density of the DSM, the number and intra-connectivity of modules, and the size of the core module. Table 1 shows initial values and ranges of each factor of the DSM used in this paper.

For case A in Table 1, in order to analyze the effect of the DSM size on modularity, we considered sizes from 50 to 950 in steps of 50 units. Case B shows the effect of the density of dependency. We assumed that the density value is 1 when every node is connected. We then changed the density from 0.05 to 0.95 in steps of 0.05 units. In this case, although edges are added at random positions, we fixed the intra-connectivity of each module to 0.8. Therefore, increasing density means increasing inter-dependencies between modules. In other words, case B is designed to demonstrate how each metric evaluates the degree of coupling between modules. Case C explains how intra-connectivity affects modularity. The intra-connectivity corresponds to the degree of cohesion in a module. Case D is useful for identifying relation between the number of modules and the modularity metrics. We increased the number of modules from 10 to 190 while fixing the density of edges. For Case E, we assumed that there is a central module which is typically called a core module or GOD module in a software system. We changed the proportion of the core module from 0.05 to 0.95 by moving nodes to the core module.

Since all of these cases generate edges at random positions, modularity results change slightly for every trial. Therefore, we prepared 100 DSMs for each case and calculated the average to achieve stable experimental results.

2.3 Measuring Results

Figure 1 presents the measured results of the metrics shown in Eqs. (1) to (4) for different modular factors. In the figure, the X-axis and the Y-axis represent the modular factor and the modularity value, respectively. Note that the modularity value of \( M_{rrc} \) should be read from the separate Y-axis on the right in each graph. The reason is that \( M_{rrc} \) exhibits very different scales and behaviors compared to the others.

Table 2 shows the Pearson product-moment correlation coefficients of the metrics for various modular factors. The correlation coefficient is 1 (or -1) in the case of a perfect in-
creasing (or decreasing) linear correlation between the variables. The values other than 1 and −1 indicate the degree of linear dependence. Table 3 presents the correlation coefficients for all pairs of the metrics. Two asterisks denote significance at the 1% level, one asterisk at the 5% level and non-asterisk at higher level than 5% in our tables which present correlation coefficients.

As shown in case A of Fig. 1, increasing the DSM size contributes to the higher modularity in $M_{new}$ and $M_{gkg}$. It turns out that the DSM size negatively correlates to the modularity in $M_{rec}$. We cannot reject the hypothesis that $M_{bunch}$ does not relate to the DSM size since its significance level is higher than 5% as shown in Table 2.

Case B shows that all the metrics experience degeneration of modularity along with increase of the edge density. In particular, Table 2 shows strong confidence for this factor. It is observed that both $M_{new}$ and $M_{bunch}$ decrease in a similar fashion. Note the edge density corresponds to the concept of coupling. It can be inferred that they share the same characteristics from the coupling perspective as we can see the maximum correlation coefficient value (case B in Table 3). The equations for $M_{gkg}$ and $M_{rec}$ are similar, too.

Case C shows that each metric increases with the increase of intra-connectivity. The intra-connectivity is related to the concept of cohesion, hence we infer that high cohesion leads to high modularity in every metric discussed here. In addition, we can see high correlation coefficients for every combination in the C rows in Tables 2 and 3. We suspect that all the metrics share similar characteristics from the cohesion perspective.

Case D shows that the larger granularity of the modules leads to the lower modularity in $M_{new}$, $M_{bunch}$, and $M_{gkg}$. In contrast, $M_{rec}$ shows jagged and positive correlations to the granularity. The case D in Table 3 shows that $M_{rec}$ is hardly related to $M_{new}$ and $M_{bunch}$ (p-value > 0.05), but it correlates to $M_{gkg}$ in a negative way. Hence we suspect that $M_{rec}$ has very different behavior on the number of modules.

Case E shows that all the metrics decrease with the increase of size of core modules. Interestingly, $M_{bunch}$ decreases steadily, whereas the other metrics plunge at some points. In addition, $M_{new}$ seems to become sensitive to size of core modules earlier than $M_{gkg}$ and $M_{rec}$.

### 3. Modularity on Evolutionary Software

This section deals with measuring modularity in open source

---

**Table 2** Correlation coefficients between modular factors and metrics.

| Factor      | $M_{new}$ | $M_{bunch}$ | $M_{gkg}$ | $M_{rec}$ |
|-------------|-----------|-------------|-----------|-----------|
| Size(A)     | 0.614**   | 0.444       | 0.678     | −0.639   |
| Density(B)  | −0.783**  | −0.783**    | −1**      | −1**     |
| Intra-Con.(C)| 0.984**   | 0.984**     | 0.993**   | 0.978**  |
| Modules(D)  | −0.830**  | −0.782**    | −0.979**  | 0.672**  |
| Core(E)     | −0.987**  | −0.989**    | −0.691**  | −0.836**|

**Table 3** Correlation coefficient of modularity metrics.

| Case       | $M_{new}$ | $M_{bunch}$ | $M_{gkg}$ | $M_{rec}$ |
|------------|-----------|-------------|-----------|-----------|
| A          | $M_{new}$ | 0.934**     | 0.975**    | −0.997**  |
|            | $M_{bunch}$ | 0.975**     | 0.831**    | −0.942**  |
|            | $M_{gkg}$ | 0.783**     | −1.000**   | 0.785**   |
|            | $M_{rec}$ | 0.785**     | 0.785**    | 1.000**   |
| B          | $M_{new}$ | 1.000**     | 0.783**    | 0.785**   |
|            | $M_{bunch}$ | 0.783**     | −1.000**   | 0.785**   |
|            | $M_{gkg}$ | 0.785**     | 0.785**    | 1.000**   |
| C          | $M_{new}$ | −1.000**    | 0.998**    | 0.984**   |
|            | $M_{bunch}$ | 0.998**     | 0.998**    | 0.986**   |
|            | $M_{gkg}$ | 0.984**     | 0.984**    | 0.984**   |
| D          | $M_{new}$ | −0.994**    | 0.779**    | −0.407    |
|            | $M_{bunch}$ | 0.779**     | 0.728**    | −0.346    |
|            | $M_{gkg}$ | 0.779**     | 0.728**    | −0.797**  |
| E          | $M_{new}$ | −0.974**    | 0.619**    | 0.781**   |
|            | $M_{bunch}$ | 0.619**     | 0.761**    | 0.898**   |
|            | $M_{gkg}$ | 0.781**     | 0.761**    | 0.926**   |

**Fig. 1** Modularity on artificial DSMs.
software by using metrics from different domains. The targeted software projects are JHotDraw (9 versions from 5.2 to 7.5), Google Web Toolkit (11 versions from 1.3 to 2.5), Apache-ant (13 versions from 1.5.2 to 1.8.4) and Linux (67 versions from 1.1.0 to 2.6.30). We extracted structural dependencies from each release by using the tool Understand [8]. We then constructed dependency graphs containing nodes for files and edges for software dependencies such as references between files, inheritance, sharing parameter, invoking method, etc. In order to modularize the graphs, we ran Community Detection clustering algorithm which is supported by Gephi [9]. Next, we generated the DSMs by using the clustering result, and measured modularity of each release. Through this process, we can detect changes of modularity in terms of software evolution. Moreover, we can check whether the modularity metrics treat the evolution in a similar way or not.

Figure 2 and Table 4 show the modularity results and the correlation coefficients. In contrast with the experimental results from artificial DSMs in the previous section, modularity graphs of open source softwares seem rather stable. We suspect that this is because the former includes drastic modular changes designed to see the characteristics of each metric, whereas the latter is from the actual changes in practice.

As shown in Fig. 2, all the metrics increase gradually with the increase of the versions of JHotDraw. In Table 4, JHotDraw row tells us that $M_{newm}$, $M_{bunch}$ and $M_{rcc}$ are positively correlated to each other, while $M_{gkg}$ is correlated only to $M_{rcc}$. In case of Google Web Toolkit, $M_{newm}$ and $M_{bunch}$ are highly correlated in a positive way. $M_{gkg}$ seems to have no correlation to other metrics. $M_{rcc}$ is negatively correlated to $M_{newm}$ and $M_{bunch}$. On the contrary, in Apache-ant project, we found correlations for the pairs of ($M_{bunch}$, $M_{gkg}$), ($M_{bunch}$, $M_{rcc}$), and ($M_{gkg}$, $M_{rcc}$). The results from Linux in Fig. 2 show that $M_{newm}$ and $M_{bunch}$ are very similar. Indeed, their correlation coefficient is 0.783 (Linux in Table 4). In addition, $M_{gkg}$ and $M_{bunch}$ are similar and their correlation coefficient is 0.670. Although these correlation values are lower than those observed in our previous experiments with the artificial DSMs, we find that this is a convincing result considering the large number of versions of Linux we used for the experiment. Note that all the changes of $M_{rcc}$ in each graph are actually very small since $M_{rcc}$ should be read from the secondary Y-axis on the right. This behavior is already observed in the previous experiments with artificial DSMs. We should be careful in applying $M_{rcc}$ in the same vein with the other metrics since $M_{rcc}$ has a unique perspective on modularity.

4. Conclusion

We compared various modularity metrics to check their ap-
plicability to evolutionary software analysis. The metrics are, in general, highly correlated; however, some metrics provide unique perspectives on particular modular factors. In open-source projects based on experiments, the degree of software evolution is not extreme as it is in the case of artificial DSMs, but a gradual change is observed. We found that modularity metrics from different domains have generally similar concepts, so it is feasible to apply the metrics for evaluating the evolutionary software. In conclusion, to evaluate evolutionary software, we need to consider comprehensive metrics and should not rely on a single metric, because a certain metric solely cannot cover the wide spectrum of perspectives on modularity. In future work, we plan to investigate other factors influencing the software modularity and consider more software projects.

References

[1] M.E.J. Newman, “Modularity and community structure in networks,” Proc. NAS of USA, pp.8577–8582, 2006.

[2] S. Mancoridis, B.S. Mitchell, Y. Chen, and E.R. Gansner, “Bunch: A clustering tool for the recovery and maintenance of software system structures,” Proc. IEEE ICSM, pp.50–59, 1999.

[3] F. Guo and J.K. Gershenson, “A Comparison of Modular Product Design Methods on Improvement and Iteration,” Proc. ASME, 2004.

[4] 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, 2006.

[5] A. MacCormack, J. Rusnak, and C. Baldwin, “Exploring the structure of complex software designs: An empirical study of open source and proprietary code,” Management Science, vol.52, no.7, pp.1015–1030, 2006.

[6] R. Miley, S. Muegge, and M. Weiss, “Design evolution of an open source project using an improved modularity metric,” Proc. IFIP, pp.20–33, 2009.

[7] N. Chiriac, K. Hölttä-Otto, D. Iysy, and E.S. Suh, “Level of modularity and different level of system granularity,” Journal of Mechanical Design, vol.133, no.10, 101007, 2011.

[8] Understand, http://www.scitools.com

[9] Gephi, https://hephi.org