ICIC Express Letters
Volume **10**, Number **8**, August **2016**

 ISSN 1881-803X
pp. **1813–1818**

A PROPOSED PACKAGE STRUCTURE AND METRICS FOR ASSESSING THE RELATIONSHIP OF COUPLING AND COHESION

Hongyan Yao and Xuebo Sun

School of Software
University of Science and Technology Liaoning
No. 185, Qianshan Middle Road, Lishan District, Anshan 114051, P. R. China
{ abroat; pupoly }@163.com

Received December 2015; accepted March 2016

Abstract. *Previous works on object-oriented software metrics are mainly focused on the issue of characterizing the class design. Although classes are important for building up software system, it is still trivial in using metrics for classes to evaluate the quality of system. To our knowledge, there are few metrics in the literature devoted to assessing packages, and also there is lack of related works to analyze those metrics necessity and the semantic behind them. In this paper we take the achievements of Robert C. Martin and Stéphane Ducasse as a basis and propose an ideal package structure as well as define the metrics for assessing the Coupling and Cohesion of it. Compared with Ducasse's metrics, ours unifies the concept of dependency and no longer makes the distinction between "Use" and "Extend"; furthermore, the defined metrics are more concise in semantic and fit well for our proposed package structure.*
Keywords: Package, Object-oriented, Metrics, Coupling, Cohesion

1. **Introduction.** Packages are not simply the class container but also they play the role of modules, which developer designs for and maintains to [1,2]. As the package could organize classes that hide intra-classes dependency, and provide concise and well identified services for the rest of the system, it becomes important to assess if the packages are well designed or still keep high design quality after decayed or refactored over years.

With regard to the packages and their structures, i.e., architecture, there are some guiding principles proposed by Martin and Martin [3], such as the Stable-Dependencies Principle (SDP) and the Stable-Abstractions Principle (SAP). It is no doubt that these principles are important for designing packages; but in the case of so many packages maintained or refactored from time to time, it is hard for the developers or the experts to realize consciously if current architecture state is still optimal [4,5]. As principles are just empirical [6], they cannot tell quantitatively the value for making a judgment. After all, "we can not control what we can not measure" [7]. Within this background, to improve the quality of software, assessing the package organization and relationships with metrics is required. However, to our knowledge there exist few works dealing with this aspect; the metrics achieved so far in [8-11] are mostly concerning to assessing the class instead of package. Even the representative literature [10,12] by Stéphane Ducasse also adopts the metrics for classes to make further definitions for assessing package relationships. We will explain in Section 2 that using metrics for classes is not suitable for assessing the package relationships.

The results achieved by Martin and Ducasse are our basis (detailed discussions are in 2.1), on which we propose an ideal state of package structure with regard to the principles suggested by Martin; meanwhile, we define the corresponding metrics for assessing the

package's Coupling and Cohesion relationship with regard to our proposed package structure. The proposed package structure extends Martin's principles of SDP and SAP while the defined metrics along with it complement Ducasse's work in the aspect of unifying Package Dependency.

2. **Analysis for the Metric-based Method and an Ideal Proposed Package Structure.**

2.1. **Analysis for the metric-based method.**

Using metric to assess the software system at the granularity of package has been mentioned by Robert C. Martin et al.

The metrics defined by Robert are: 1) Afferent Couplings (C_a), which means the incoming couplings; 2) Efferent Couplings (C_e), which means the outgoing couplings; 3) Instability (I), $I = C_e/(C_e + C_a)$, which is viewed as an indicator of the package's resilience to change; 4) Abstractness (A), $A = N_a/N_c$, which is to assess if the package is stable (N_c denotes all classes in the package while N_a denotes all abstract classes in package); 5) $A + I = 1$, which is called "The Main Sequence" in the A/I graph. If the perpendicular distance from the package location to "The Main Sequence" is lesser, the package is more ideal in structure, because closer to the main sequence means the package has enough abstract classes to keep its inner frame stable; meanwhile, it also indicates the package depends on few other packages, so it is easy to locate and maintain the packages where the changes break out.

Although the metrics defined by Martin could support "the Stable-Dependencies Principle (SDP)" and "the Stable-Abstractions Principle (SAP)", they are still incapable to further tell: 1) to which extent a package depends on the other? 2) are services provided by the package Cohesive? For solving these two deficits Ducasse et al. proposed several supplements in formal definition focusing on the relationships of Coupling and Cohesion. The metrics they proposed for Coupling are IIPU, IIPE, IICI, IIPUD, and IIPED [12]; for Cohesion are PF, IPSC, etc. We believe that the Coupling represents nothing but the dependency. Although dependency could be divided into two categories: Use and Extend, they are the same in nature: to reference another class (no matter what style the class is: base class, regular class, interface class, or abstract class). So the metrics like IIPU and IIPE, dealing with Use dependency and Extend dependency respectively, did not get as much senses as we thought. Another thing we noticed is that according to the value of PF and IPSC how to adjust the set of services was not mentioned in [10,12], which is actually important for improving the package Cohesion.

2.2. **A proposed package structure.**

Following Martin's options for the package, the package structure we proposed, shown in Figure 1(a), expands and groups the package elements into 5 terminologies: Factory Pattern, Frame Class, Abstract Class, Interface Class, and Regular Class. The interpretation for Figure 1(a) is this: inner four categorized classes mixed together for providing services through the declaration of In/Out Interfaces. The role for each inner four classes is interpreted as this: 1) Abstract Class has the role of base. It provides the base for the whole package, implementing partially the Interface Classes or providing the basis inherited by Regular Classes. In future the Abstract Classes may be extended to support the refactored Interface Classes. 2) Interface Class has the role of declaration. Interface means nothing but to tell inner or outer space the interface type and Functions Signature that may be referenced. You can revise the interface anyway only if two parties agree with it. 3) Regular Class has the role of encapsulating. Derived from Abstract Classes, implementing necessary Protected/Private methods is its job; besides, regular class object is the product of Factory Pattern. 4) Frame Class has the role of resource manager, which provides the data or cooperative objects needed by Regular Class Object. Frame Class is usually concrete but derived from Abstract Class, aiming for using DIP.

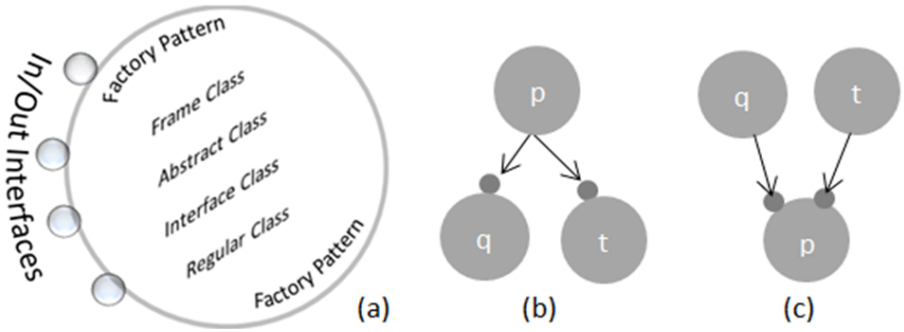

FIGURE 1. The ideal package structure and two examples for package coupling

Pattern Factory in Figure 1(a) does one work: to provide the objects that support Out-Interfaces, i.e., provide object that knows well certain Out-Interface. As far as the In-Interfaces are concerned, they only tell the other packages that In-Interfaces will be used in the scope of this package, and if certain In-Interface is indeed used, a corresponding object should be passed on at the time. In-Interfaces sometimes could be viewed as Events.

3. **Formal Metric Definitions for Assessing the Coupling and Cohesion.** Next we present the metrics for assessing the Coupling and Cohesion with regard to Figure 1(a).

3.1. **Preliminary notations.** An object-oriented software system is defined by $M =< P, D >$. P denotes all packages and D denotes pairwise dependencies among the packages, $D \in P \times P$. The set of I denotes all In/Out Interfaces that belong to each package, noted as $I = Int(P) \cdot (InInt(p) \cup OutInt(p)) \subseteq Int(P)$, $p \in P$. Predicate $Dep(i_1, i_2)$ is true if i_1 depends on i_2, and i denotes an interface; similarly, $Dep(p_1, p_2)$ is true if p_1 depends on p_2, and p denotes a package. For convenience, one argument version of $Dep(i)$ denotes all interfaces that i depends on; $Dep(p)$ denotes all packages p depends on. $\text{Clients}_p(p)$ denotes the packages which depend on p while $\text{Providers}_p(p)$ denotes the packages p depends on. Predicate $P(i)$ denotes the package where i belongs to.

3.2. **Coupling metrics.** *Index of Inter-Package Dependency (IIPD).* See Definition 3.1. Its value ranges from 0 to 1 ($IIPD = 1$ if denominator is 0). The package structure is better when *IIPD* is closer to 1, because it reflects the facts that upper level package depends on few interfaces provided by lower packages. For instance, see Figure 1(b). p depends on two interfaces packaged into q and t respectively, and then $IIPD = 1$. Otherwise, $IIPD < 1$ if p depends on more interfaces that belong to q or t. A lesser *IIPD* value hints that, on one hand, there is a package that depends on many interfaces and this leads to the package instability; on the other hand, the interfaces that belong to $\text{Providers}_p(p)$ were not focused but segregated.

Definition 3.1.

$$IIPD(M) = DepSum(P)/DepSum(I)$$

$$DepSum(I) = \sum\nolimits_{i \in I} |Dep(i)|; \quad DepSum(P) = \sum\nolimits_{P_j \in P} \sum\nolimits_{i_k \in P_j} |ExternalDep(i_k)|$$

$$ExternalDep(i) = \{x | Dep(i, x) \& P(i) \neq P(x) \& x \in I\}$$

Index of Inter-Package Changing Impact (IPCI). As our package structure has the surface of interface, the changing impact is only caused by Out-Interface which is responsible for providing services. The changed In-Interfaces only influence the package itself, not the other packages, i.e., changing impact caused by In-Interfaces is not spread out. See Definition 3.2. *IPCI* takes the value of 0 to 1. Changing impact is less if the value is smaller. For instance, see Figure 1(c). If there are all three packages in M, then $IPCI(p)$

will be 1, which means it will influence all the other packages if p is changed; if M contains more than 3 packages, then $IPCI(p) < 1$. A greater value of $IPCI(M)$ hints that there is package that supports too many interfaces, and that will lead to: on one hand, the package is fat in the Out-Interfaces and the package goal is unfocused; on the other hand, the package is inflexible to adjust.

Definition 3.2.

$$IPCI(p) = |Client_p(p)|/(|P| - 1);$$
$$IPCI(M) = \sum\nolimits_{P_i \in P} IPCI(p_i)/(|P| - 1)$$

Index of Inter-Package Dependency Diversion (IPDD). If a package p depends on 4 interfaces organized into 4 different packages, we call p dependency is totally diverted; otherwise, focused if 4 interfaces are organized into 1 package. For measuring the extent to which p dependency is diverted, metric of *IPDD* is defined. See Definition 3.3.

Definition 3.3.

$$IPDD(p) = PDF(p) \times (|Providers_p(p)|/|Providers_I(p)|)$$
$$Providers_I(p) = \{ExternalDep(x)|x \in InInt(p)\}; \quad PDF(p) = 1/|Dep(p)|$$

For instance, if p depends on 4 packages then $PDF(p) = 1/4$. However, if p depends on 7 interfaces scattered into this 4 packages, the *IPDD* will be $(1/4)\times(4/7)$. If p depends on more than 7 interfaces, *IPDD* will be much lesser. A lesser value of *IPDD* will hint us; on one hand, p has more dependencies on other packages; on the other hand, the interface set of certain provider package should be reconsidered to split according to its responsibility because the goal of its services is not focused.

3.3. **Cohesion metrics.** With regard to the proposed package structure, we define two metrics for measuring cohesion. The first, *PRF*, is to assess if the interfaces belonging to a service are commonly used bindingly; the second, *PSC*, is to assess if the interfaces are organized reasonably for providing several services.

Index of Package Role Focus (PRF). In ideal state, package should focus on providing one well identified service to the rest of the software system. We call a package providing focused service if it plays the same role with all its clients. Let $ReqInt(p, q)$ denote p interfaces that are required by q, and then the role of p is defined in Definition 3.4 as $PRF(p)$ for measurement.

Definition 3.4.

$$PRF(p) = \sum\nolimits_{p_i \in Clients_p(p)} Role(p, p_i)/|Clients_p(p)|$$
$$ReqInt(p, q) = \{i \in OutInt(p)|\forall x \in InInt(q) \& q \in Clients_p(P) : Dep(x, i)\}$$
$$Role(p, q) = |ReqInt(p, q)/InInt(p)|$$

PRF takes its value between 0 and 1 and it is straight forward in semantic that 1 is the value we expected, which indicates all p Out-Interfaces are either used all together or not used at all. However, that is not the usual case. In general, the higher value $PRF(p)$ has, the higher frequency of requiring largest portion of p Out-Interfaces. Higher value of $PRF(p)$ will hint us to mask some uncommonly used or deprecated interfaces.

Index of Package Services Cohesion (PSC). If a package p provides more services, a question must be addressed: are the interfaces organized corresponding to each service reasonably? For answering this question, we define the metric of *PSC*, see Definition 3.5. Let $CS(p, q)$ denote the Composite Services provided by p to q, $CS(p, q) = \{x \in OutInt(p)|x \in InInt(q)\}$. Let $SM_k(p, q) = |CS(p, q) \cap CS(p, k)|/|CS(p, q)|$ denote the metric of similarity for different two clients of p. $SM_k(p, q) = 1$ when numerator is 0.

Definition 3.5.

$$PSC(p) = \sum\nolimits_{q_i \in Clients_p(p)} SM_{Cohesion}(p, q_i)/|Clients_p(p)|$$

$$SM_{Cohesion}(p, q) = \sum\nolimits_{k_j \in Clients_p(p)} SM_{k_j}(p, q)/|Clients_p(p)|$$

The value of SM is between 0 and 1, the same as PSC. 1 being the value optimal $SM_{Cohesion}(p, q) = 1$ means p is completely cohesive, because the service, the set of interfaces provided by p to q, is just the same one that is provided to the other clients. The closer is the value of SM to 1, the more cohesive the p's service is. Likewise, if the $PSC(p)$ is larger and close to 1, then each of services provided by p is more cohesive. Otherwise, the interface set of certain service should be re-organized.

4. **Comparison with Ducasse's Work.** We will compare ours with Ducasse's work in these three aspects. See Table 1.

Table 1. A comparison between our proposed metrics and Ducasse's

Metrics proposed by	Make distinction between "Use" and "Extend"?	Calculate metrics with package inner class?	Build metrics on interfaces?	On what level to adjust for satisfying requirement change?	Propose a matched package structure?
Ducasse's	Yes	Yes	No	Inner classes	No
Ours	No	No	Yes	Interface	Yes

1) We do not make the distinction between "Use" and "Extend". In Ducasse's they defined different metrics for measuring Use dependency and Extend dependency. We thought this is unnecessary because according to Martin's point of view any Extend dependency could be changed reversely by using interface and Dependency-inversion Principle. That means Extend is also a kind of "Use" after transformation. We agreed to this. So in our metrics, we differentiate Extend from Use no more. Only use predicate *Dep* to indicate the dependency relationship between packages.
2) We do not define the package metrics by considering class metrics. In Ducasse's they defined package metrics by introducing class metrics. Especially, the Intra-classes dependency is accounted for defining package metrics, and this we thought is inappropriate. We believe no matter how complex the package inner structure is, it should not be accounted for defining metrics that are used to assess package exposed services and the relationships between all packages. So in our package metrics definition we only build them on the package interfaces.
3) As the interfaces exposed by package are relatively lesser than the classes that package contained, it is more controllable to handle the interfaces dependency issue if the values of metrics for Coupling or Cohesion is not reasonable. After all, interface declaration can isolate itself from package inner implementation, and simply adjusting the package interfaces could increase the package design efficiency a lot. However, in Ducasse's metrics, they did not mention the classes exposed by a package are in which stereotype: abstract, concrete, or interface? If it is concrete, then the dependency adjustment will cause a lot more works than adjusting interface. Because as Martin indicated: to extend the abstractive is more effective in the long run than modifying the concrete.

5. **Conclusion.** As the packages are the units of software system, it is important to define metrics to assess the relationship between them: Coupling and Cohesion. However, the results achieved so far mainly focus on the metrics for measuring classes; few works proposed metrics for assessing packages relationship. We borrow the thought of Martin and Ducasse's and propose an ideal package structure as well as metrics for assessing

Coupling and Cohesion. The package structure of our work extends Martin's principles of SDP and SAP, and the metrics along with it complement Ducasse's work in unifying Package Dependency, removing intra-classes dependency, and simplifying metrics definition expression.

REFERENCES

[1] H. Yao and Y. Ma, An exploration for the software architecture description language of WRIGHT, *ICIC Express Letters*, vol.8, no.12, pp.3481-3487, 2014.

[2] H. Yao, Y. Jiang and W. Shen, A revised orthogonal software architecture COA, *ICIC Express Letters*, vol.7, no.8, pp.2255-2261, 2013.

[3] R. C. Martin and M. Martin, *Agile Principles, Patterns, and Practices in C#*, Prentice Hall, 2008.

[4] L. M. Hakik and R. E. Harti, Measuring coupling and cohesion to evaluate the quality of a remodularized software architecture result of an approach based on formal concept analysis, *International Journal of Computer Science and Network Security*, vol.14, no.1, pp.11-16, 2014.

[5] P. Gandhi and P. K. Bhatia, Optimization of object-oriented design using coupling metrics, *International Journal of Computer Applications*, vol.27, no.10, 2011.

[6] J.-R. Falleri, S. Denier, J. Laval et al., Efficient retrieval and ranking of undesired package cycles in large software systems, *Objects, Models, Components, Patterns – The 49th International Conference, TOOLS*, Zurich, Switzerland, 2011.

[7] H. Abdeen, S. Ducasse and H. Sahraoui, *Modularization Metrics: Assessing Package Organization in Legacy Large Object-Oriented Software*, https://hal.inria.fr/inria-00614583, 2011.

[8] L. C. Briand, J. W. Daly and J. K. Wüst, A unified framework for coupling measurement in object-oriented systems, *IEEE Trans. Software Engineering*, vol.25, no.1, pp.91-121, 1999.

[9] S. A. Ebad and M. A. Ahmed, Functionality-based software packaging using sequence diagrams, *Software Quality Journal*, vol.23, no.3, pp.453-481, 2015.

[10] J. Laval and S. Ducasse, Resolving cyclic dependencies between packages with enriched dependency structural matrix, *Software: Practice and Experience*, vol.44, pp.235-257, 2014.

[11] G. Santos, N. Anquetil, A. Etien and S. Ducasse, System specific, source code transformations, *The 31st IEEE International Conference on Software Maintenance and Evolution*, Nicolas Anquetil, 2015.

[12] S. Ducasse, N. Anquetil and U. Bhatti, Software metrics for package remodularisation, *HAL – INRIA*, 2011.