Revisiting The Package-level Cohesion Approaches

Waleed Albattah
Information Technology Department
Qassim University
Qassim, Saudi Arabia
e-mail: w.albattah@qu.edu.sa

Suliman Alsuhibany
Computer Science Department
Qassim University
Qassim, Saudi Arabia
e-mail: salsuhibany@qu.edu.sa

***Abstract*—Software measurements play a critical role in assessing software properties. Cohesion is one of the software properties that are considered to have a relationship with software quality. Many cohesion metrics have been proposed by researchers to assess cohesion on different software abstractions, i.e., class-level and package-level. The proposed package-level cohesion metrics in the literature seem to differ in their assessment of cohesion. In this paper, we try to investigate this issue and establish whether cohesion has only one concept. The conclusion of this paper encourages further investigation and comparison between the existing package-level cohesion metrics.**

Keywords—Cohesion; package; metric; measurement; software.

I. INTRODUCTION

With the increased importance of software measurements in assessing software properties, research works have produced and are continuing to produce new software measures. One specific type of measure is cohesion. Cohesion refers to the degree to which the elements of a specific component belong together [3].

During software maintenance, developers spend at least 50% of their time analysing and understanding software [2]. In object-oriented programming languages, e.g., Java, assembling only closely related classes into packages can improve software maintenance. Package cohesion metrics measure the coherence of a package amongst its elements that should be closely related. Cohesion is an internal attribute of software that affects its maintainability and reusability. Following the design principles [21], a high level of cohesion has as its goal to achieve software maintainability and promote its reusability [22][26].

Package-level cohesion research has received very little focus compared with research on other abstractions, e.g., class-level. When one examines the literature on package cohesion metrics, it is clear that there are significant differences in these metrics. Thus, the following natural question arises: do these metrics measure the same thing? This question will be addressed in this paper.

The paper is organised as follows. In Section II, we present Package Cohesion Principles [21]. The existing approaches to package cohesion are presented in Section III. Section IV presents the general example for all the existing approaches. The conclusion and future work are given in Section V.

II. PACKAGE COHESION PRINCIPLES

R. C. Martin [21] has presented six principles for package design, which have since become well-known and well-accepted. The first three principles are for package cohesion and they help to allocate system classes to packages. This allocation can help to manage the software during its development. In our previous work [23], the three package cohesion principles of Martin [21] were discussed and they are introduced here briefly from [23]:

1) The Reuse-Release Equivalence Principle (REP)
"The granule of reuse is the granule of release"

This states that the reuse of the code should be the same size as the release one. If a person decides to reuse someone else's code, he needs a guarantee that the support will continue and the release of new versions will be on the same original size. To ensure the reusability of the code, the author must organise the classes into reusable packages and then track them with the release.

2) The Common Reuse Principle (CRP)
"The classes in a package are reused together. If you reuse one of the classes in a package, you reuse them all"

This principle tells us which classes should be grouped together. As it states, the classes that tend to be reused together should be in the same package. It is more likely for reusable classes to depend on each other, so classes are rarely reused in separation. CRP states that the classes of a package should be inseparable, which means that if a package depends on this package, it should depend on all of its classes and not on a number of them. In short, classes that are not tightly coupled to each other should not be kept in the same package.

3) The Common Closure Principle (CCP)
"The classes in a package should be closed together against the same kinds of changes. A change that affects a package affects all the classes in that package and no other packages"

From the maintenance point of view, while the change is not avoidable, it should be controlled (minimised). If a change has been made on one package, there is no need to

 ISBN: 978-1-61208-438-1

re-release or revalidate packages that do not depend on the changed package. The CCP states that the classes in the package should not have different reasons to change.

While the previous two principles, REP and CRP, focus on reusability, the CCP focuses on the system maintainability. If a change is made on the code, it would be better to be on one package or on a few packages rather than being on many packages. The classes that are tightly related will change together. Hence, if they are kept in the same package, only one package or a small number of packages are going to be affected when a change happens. Also, the effort regarding revalidating and re-releasing of software will be minimised.

III. THE EXISTING PACKAGE COHESION APPROACHES

A number of cohesion approaches have been proposed on class and method levels [1][3]-[6]-[18]. In this section, we present some of the existing package-level cohesion approaches. A brief description is given for each. In the literature, Misic [19], Ponisio and Nierstrasz [22], Martin [21], Xu et al. [20], Zhou et al. [24], Abdeen et al. [25], and Albattah and Melton [23] have each proposed different methods to measure package cohesion. Each proposes a cohesion metric on the package level. A brief discussion for each approach is given next.

A. Approach by Misic

Misic [19] proposes a way to measure a functional cohesion. Since a number of approaches were focusing on cohesion as an internal structure issue, Misic claimed that cohesion could be also observed externally by focusing on its functional property regardless of the package's internal structure.

The approach measures the similarity of package objects (elements). The similarity between elements can be measured by looking at the external clients' usage patterns.

Method

Misic defined write and read range concepts. The write range of an object *O*, *W(O)*, refers to the set of objects (servers) used by this object (client). The read range of an object *O*, *R(O)*, refers to the set of objects (clients) used by this object (server).

Given a set of objects *S*, let *R(S)* be its client set (Read range), S_w the subset that IS? used to write its clients, and let $S_w(x)$ be the part of that subset that IS? used to write the client *x*. Then, the coherence is given by the following formula:

$$\psi(S) = \frac{\sum_{x \in R(S)} (\#S_w(x) - 1)}{\sum_{x \in R(S)} (\#S - 1)} \quad (1)$$

where

#*S* stands for the number of elements in *S*.

The coherence measure proposed by Misic can be calculated internally or externally. For internal coherence, the summation in the numerator and denominator will be restricted only for clients inside the questioned set. Similarly, the summation will be restricted only for clients outside the questioned set to measure the external cohesion.

B. Approach by Ponisio and Nierstrasz

Ponisio and Nierstrasz [22] proposed a similar approach to measure package cohesion. The proposed contextual metric measures the cohesion based on the common use by clients. The approach idea is to propose the Common-Use (*CU*) metric that measures the package cohesion by taking into account the way that a package's classes are accessed by other packages.

Method

CU measures the cohesion of package *P* by considering the use of its elements by the package clients. If all the clients use the same set of *P's* elements, these elements share the same responsibilities of *P*, and then *P* is cohesive. Instead, if the clients use a different set of *P's* elements, these elements have different responsibilities, and then *P* is not cohesive.

There is a need for weight to differentiate between client packages. Not all clients have the same degree in assessing *P's* cohesion. The weight reduces the influence of *P's* cohesion from the promiscuous clients.

Definition: "*We define the weight of a (client) package* P_{client} *as the inverse of the number of connections that* P_{client} *has with other packages."*

$$w(P_{client}) = \frac{1}{fan\ in(P_{client}) + fan\ out(P_{client})} \quad (2)$$

The definition of *CU* is given as follows:

"*We define Common-Use (CU) as the sum of weighted pairs of classes from the interface of a package having a common client package (f), divided by the number of pairs that can be formed with all classes in the interface.*"

$$\mathsf{CU} = \sum_{a,b \in I} \frac{\mathsf{f}(a,b) * weight(a,b)}{\#Pairs} \quad (3)$$

Where

$$\begin{aligned} I &= \mathsf{interface}(P) \\ \#Pairs &= \tfrac{|I| \times (|I|-1)}{2} \\ C &= \mathsf{clients}(a) \cap \mathsf{clients}(b) \\ \mathsf{f}(a,b) &= \begin{cases} 1, & if\ C \neq \emptyset \\ 0, & otherwise \end{cases} \\ weight(a,b) &= \textstyle\sum_{c \in C} \frac{w(c)}{|C|} \end{aligned}$$

The value of *CU* is between 0, which represents that the interface classes of the package have disjoint responsibilities, and 1, which means that the interface classes of the package are used together.

C. Approach by Martin

Martin [21] presents a set of principles of object-oriented package design. Three of these principles, package cohesion principles, try to help the software architect to organise classes over packages. These principles are: REP, CCP, and CRP, discussed earlier in Section II. The three principles aim to provide a high quality of package cohesion.

Method

Martin [21] proposed a number of simple package-level metrics. One of them is a relational cohesion of a package. The package cohesion metric is presented as an average number of internal relations per class. Regardless of the package external dependencies that are considered in other cohesion metrics, the metric measures the connectivity between package elements. This metric is quite simple to apply, and is given by:

$$H=(R+1)/N \quad (4)$$

where

H: package cohesion
R: number of internal relations
N: number of the package classes

The extra "1" in the numerator prevents cohesion *H* from equalling zero when *N*=1. This metric gives all internal relations the same weight and disregards the external ones. It has been applied to a number of software projects and is widely accepted.

D. Approach by Xu et al.

Xu et al. [20] propose an approach to measuring the package cohesion in Ada95 object-oriented programming language. The proposed metric is based on dependence analysis between package entities. It is assumed that the package may have objects and sub-programs.

Method

The package dependence graph (*PGDG*) describes all types of dependencies: inter-object dependence graph (*OOG*), inter-subprogram dependence graph (*PPG*), and subprogram-object dependence graph (*POG*). The method measures package cohesion according to *PGDG*. It assumes that package *PG* has *n* objects and m subprograms, where *n*, $m > 0$.

To present the measure in a unified model, a power for each object *PW(O)* is given:

$$\begin{cases} Cohesion(O) & O \text{ is a package object} \\ Cohesioin(PG(O)) & O \text{ is a type object} \\ 1 & others \end{cases}$$

Xu et al. [20] claimed that, according to the definitions, it is easy to prove that the measure satisfies the four properties given by Briand et al. [3][27] to develop a good cohesion measure.

However, an Ada package represents a logical grouping of declarations. The role of an Ada package is similar to the role of class in other languages, such as Java [24]. Thus, this package cohesion metric cannot be applied to the general example in the next section. An Ada package actually falls in the category of class-level cohesion metric.

E. Approach by Zhou et al.

Zhou et al. [24] proposed an approach to measuring package semantic cohesion called the Similar Context Cohesiveness (*SCC*). In this approach, the common context is used to assess the degree of relation between two components. *SCC* measures the inter- and intra-package dependencies that can reveal semantic cohesion between components.

Method

The proposed package cohesion measure *SCC* is based on the component context. The context of component *c* is composed of two sets: the components that depend on *c* and those that *c* depends on. The *SCC* metric is given by:

$$SCC(p) = \begin{cases} \dfrac{\sum_{(c_i,c_j)\in E(p)} Wgt(c_i,c_j)}{m(m-1)} & if \quad m>1 \\ 1 & if \quad m=1 \end{cases} \quad (5)$$

where

m: number of components c in p

$\mathrm{Wgt}(c_1, c_2) = \mathrm{CCS}(c_1, c_2) + \mathrm{Dep}(c_1, c_2)$

$$Dep(c_1,c_2) = \begin{cases} 1 & if \quad c_1 \xrightarrow{d} c_2 \quad or \quad c_2 \xrightarrow{d} c_1 \\ 0 & else \end{cases}$$

$CCS(c_1,c_2)$: denotes the similarity between the contexts of two components c_1 and c_2 , and is given by:

$CCS(c_1, c_2) = kRSS(c_1, c_2) + (1\text{-}k)DSS(c_1, c_2)$

k: represents the position's importance

RSS(c1, *c2*): similarity between $S_R(c1)$ and $S_R(c2)$

DSS(c1, *c2*): similarity between $S_D(c1)$ and $S_D(c2)$

$$S_R(c) = \{c_i \mid c_i \xrightarrow{d+} c\}$$

$$S_D(c) = \{c_i \mid c \xrightarrow{d+} c_i\}$$

F. Approach by Abdeen et al.

The approach proposed by Abdeen et al. [25] is based on the Simulated Annealing technique. The approach aims to reduce package coupling and cycles by moving classes between packages. Two metrics have been defined for this purpose, coupling and cohesion metrics.

Method

The approach automatically reduces package coupling and cycles by moving classes between packages considering the existing class organisation and package structure. This approach can help maintainers to define: the maximum number of classes that can change their packages, the maximum number of classes that a package can contain, and

the classes that should not change their packages and/or the packages that should not be changed. A set of measures is defined to determine and quantify the quality of a package. The number of package dependencies ($|P_D|$) normalises these measures.

The package cohesion metric is defined to be the direct dependencies between its classes. Hence, the cohesion of a package P is proportional to the number of its internal dependencies ($|P_{Int.D}|$) according to the CCP Principle [19]. The cohesion quality is given as follows:

$$CohesionQ(p) = \frac{|p_{Int.D}|}{|p_D|} \quad (6)$$

where

$|P_D|$ is the number of all internal and external dependencies of classes in the package.

G. Approach by Bauer and Trifu

Bauer and Trifu [28] have proposed an approach, architecture-aware adaptive clustering, to produce meaningful decompositions in a system. They have evaluated their approach by defining two metrics: the average cohesion of a subsystem and the average coupling between subsystems.

Method

The approach was based on providing better understanding of the system. They tried to recover from the original decomposition and then impose an appropriate structure. The new structure aims to maximise subsystems cohesion. To evaluate the recovered subsystem decomposition, they performed a comparative study that is based on two criteria, accuracy and optimality. For accuracy, they compared the resulting decompositions with both the original package structure and the ideal Common Reuse Principle structure of [21]. For optimality, they used some optimality metrics to show whether the resulting decompositions have high cohesion and low coupling. To evaluate their approach, two metrics were defined: average cohesion of the subsystems and average coupling between the subsystems of a given decomposition. The average cohesion metric is given by:

$$avgCohesion(D) = \frac{\sum_{\substack{S_i \in D \\ |S_i|>1}} \frac{noInternalEdges(S_i)}{\frac{|S_i|^2-|S_i|}{2}}}{|D|^*} \quad (8)$$

where

D: a composition

noInternalEdges(Si): number of edges between classes in *Si*

$|D|^*$: number of subsystems except single-class subsystems in D

Si: subsystem number *i* in *D*

$|Si|$: number of classes in subsystem *Si*

H. Approach by Seng et al.

The approach by Seng et al. [29] aims to develop existing object-oriented system decompositions by defining new decompositions with better metric values and fewer violations of design principles. They defined the problem as a search problem. The quality of the resulting subsystem decompositions is measured by the fitness function that combines software metrics and design heuristics.

Method

The fitness function consists of cohesion, coupling, complexity metrics, as well as cyclic dependencies and bottleneck heuristics. The value of each individual function is between 0 and 1, where the optimal value is 1. The cohesion of a system s is the summation of cohesion values for the individual subsystems in s. The cohesion for a subsystem s_i is measured by counting the number of different classes in s_i known by some class $c_j \subset s_i$,(#k(c_j)), and dividing this by the square number of classes in s_i , (#c(s_i)). The resulting value can be normalised if divided by the number of subsystems (#s).

$$cohesion(s) \quad : \quad \frac{\sum_{i=1}^{\#s} \sum_{j=1}^{\#c(s_i)} \frac{\#k(c_j)}{\#c(s_i)^2}}{\#s} \quad (7)$$

I. Approach by Tagoug

Tagoug [30] has proposed coupling and cohesion metrics on subjects, which are similar to packages. Each subject is a collection of classes. The approach aims to measure cohesion and coupling at the system level. The quality metric, which combines cohesion and coupling values, measures the decomposition's quality as early as the analysis and design phases of the software development lifecycle.

Method

The two metrics measure the quality of object-oriented decomposition. The cohesion metric focuses on the interactions of components inside a subject, while the coupling metric focuses on the interactions of components among subjects. The cohesion of subject E is given by:

$$C(E) = \frac{\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} Wij}{W_{\max} * (n*(n-1)/2)} \quad (9)$$

where

E: a set of classes of S.
Wij: the sum of the weights of links in *Lij*.
Lij: the set of all links between classes *Pi* and *Pj*.
Wmax = max $\{Wij\}$ in system S

$n = |E|, n > 1$

The cohesion value is between 0, i.e., there are no links between classes, and 1, maximum links with maximum weight. The weights of links between classes of a subject are ordered in Table I based on the degree of association according to the object-oriented expert designers.

TABLE I. WEIGHTS OF LINKS BETWEEN CLASSES.

Links Type	Weights (*Wij*)
Whole Part Structure	0.9
Inheritance	0.8
Instance Connection	0.7
Message Connection	0.6
Conceptual Link	0.5

J. Approach by Albattah and Melton

The approach by Albattah and Melton [23] is motivated by the package cohesion principles [21]. They proposed two different cohesion metrics to measure two different cohesion concepts or types based on Martin's package cohesion principles in [21]. The first cohesion type, Common Reuse (*CR*), includes the factors that help in assessing *CR* cohesion. Similarly, the second cohesion type, Common Closure (*CC*), includes the factors that help in assessing *CC* cohesion. After each type of cohesion is measured by itself, the two values of *CR* and *CC* may be combined to one unified value of package cohesion, while still recognising the two types.

Method

The *CR* metric measures cohesion based only on the common reuse factors of the package. The elements of a package have different degrees of reachability. Reachability of a class in a package is the number of classes in the same package that can be reached directly or indirectly. The *CR* metric is defined as follows:

"Let $c \in C$, and suppose there is an incoming relation to *c* from a class in a different package. Then *c* is called an in-interface class. The cardinality of the intersection of the hub sets of all the in-interface classes in *C* divided by the number of classes in *C* is the *CR* of *P* ".

$$CR = |\cap_{\text{In-interface class hub sets}}| / |C| \quad (10)$$

where

Hubness(c) = {d ∈ C: if there is a path c → d}

C: set of classes in package *P*

c and d: classes in C

The *CC* metric considers the package dependencies on other packages as well as the internal dependencies between classes of the package. The classes of the package should depend on the same set of packages and, thus, they will have the same reasons for a change. The *CC* metric is defined as follows:

"The cardinality of the intersection of the reachable sets divided by the cardinality of the union of the sets represents the *CC* of *P* ".

$$CC = (|\cap_{\text{Reachable Package sets}}| / |\cup_{\text{Reachable Package sets}}|) \quad (11)$$

The combined cohesion *CH* is defined as follows:

$$CH = \frac{\sqrt{2} - D}{\sqrt{2}} \quad (12)$$

$$D = \sqrt{(1-CR)^2 + (1-CC)^2} \quad (13)$$

IV. THE GENERAL EXAMPLE

While we try to understand each of the previously presented approaches, we rely on our best understanding for each. One method of empirical investigation is to apply all the approaches on the same situation and compare the results. The approaches have been applied to measure the cohesion of *P1* in Figure 1. The concern is to measure the cohesion of *P1* only for the purpose of comparison between the approaches. If all the approaches rely on the same idea, their assessments of the cohesion of *P1* will be alike. Otherwise, they probably rely on different concepts of package cohesion.

In Figure 1, there are six packages and a number of classes in each package. The arrows represent the dependencies between classes within the same package, i.e., in *P1*, or between classes in different packages. The direction of the dependency is very important because it shows the depended-upon class. For example, *C6* depends on *C2* but not the opposite. In the figure, *P1* has four classes that have incoming and outgoing dependencies. Using the presented approaches, we try to measure how cohesive are the classes of *P1*. It is worth mentioning that all the presented approaches consider the dependencies between classes to measure cohesion, but in different ways. Some approaches, such as Albattah and Melton [23], consider the direction of the dependencies. However, some other approaches, such as Martin [21], do not consider the direction of the dependencies. For this difference and other differences between the presented approaches, it is expected to find distinct cohesion assessment values for *P1*.

Again, all calculations are made based on our own understanding of each approach.

 ISBN: 978-1-61208-438-1

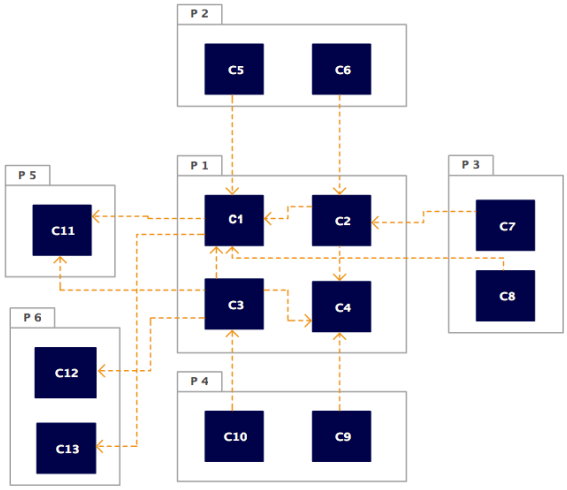

Figure 1. The general example.

Table II presents the cohesion values of package *P1* for the different approaches.

TABLE II. COHESION VALUES OF THE PRESENTED APPROACHES.

Approach	Cohesion			
	Metric	Value	Min	Max
Misic [19]	*Ψ(S)*	0.33	0	1
Ponisio and Nierstrasz [22]	*CU*	0.125	0	1
Martin [21]	*H*	1.25	> 0	N(N-1)*
Zhou et al. [24]	*SCC(p)*	0.36	0	1
Abdeen et al. [25]	*CohesionQ(p)*	0.29	0	1
Bauer and Trifu [28]	*avgCohesion(D)*	0.67	0	1
Seng et al. [29]	*cohesion(s)*	0.25	0	1
Tagoug [30]	*C(E)*	0.67 **	0	1
Albattah and Melton [23]	*CH*	0	0	1

* N: number of classes in the package

**Assuming that all the connections are instance connections with 0.7 weights.

Although all the presented approaches have the same range of cohesion values except Martin's approach [21], they end up with different cohesion values for the same package, i.e., *P1* in Figure 1. For example, the approaches by Bauer and Trifu [28] and Tagoug [30] assess the cohesion of *P1* as relatively high. In contrast, the approach by Albattah and Melton [23] assesses the cohesion of *P1* as poor.

This simple comparison raises a question about the theory behind these different approaches. The distinct evaluation results for the same package means that the presented approaches rely on different views of cohesion. These views can be noticed by investigating the presented approaches. We believe cohesion has different types or parts and some approaches focus only on one part. This can lead to misleading cohesion assessments. Cohesion has three different concepts that led to different approaches. The first concept considers cohesion as an internal property of a package that can be measured from inside the package only, such as the approach by Martin [21]. The second concept considers cohesion as a property that can be measured from outside the package, such as the approach by Ponisio and Nierstrasz [22]. The third concept considers cohesion to be measured from both inside and outside the package, such as the approach by Albattah and Melton [23].

These three concepts represent three scopes where cohesion has been measured in the presented approaches. The scope of package cohesion can be used to classify the presented approaches. Table III presents this classification based on the scope of cohesion used in each approach, i.e., internal, external, or both.

TABLE III. CLASSIFICATION OF THE PRESENTED APPROACHES.

Approach	Method	Scope of Cohesion	
		Internal	External
Misic [19]	External Objective		✓
Ponisio and Nierstrasz [22]	Common Use of the package		✓
Martin [21]	Relational Cohesion	✓	
Zhou et al. [24]	Similar Context Cohesiveness	✓	✓
Abdeen et al. [25]	Dependency Analysis	✓	
Bauer and Trifu [28]	Average Cohesion	✓	
Seng et al. [29]	Dependency Analysis	✓	
Tagoug [30]	Interactions inside the package	✓	
Albattah and Melton [23]	Common Reuse & Common Closure	✓	✓

The classification in Table III can reveal, somehow, the reason behind the diversity of package cohesion approaches that led to distinct results in Table II. Package cohesion has been viewed in different ways. It is worth saying that all the views may be right but they are different. This leads to the idea that there is more than one type of cohesion. The previous research works treated cohesion as one type or one concept, except for the research carried out by Albattah and Melton [23], and this was not accurate in our opinion.

We support the idea of Albattah and Melton [23] that is presented in this paper about cohesion. They defined cohesion as an internal property of the package and it has two different types. The first type can be measured from outside the package and it represents how well the classes in

 ISBN: 978-1-61208-438-1

the package cooperate to provide a service to classes outside the package. The second type measures how well the classes in the package are closed in using classes in other packages. This type represents the closure of the package's classes against the same kind of changes, which is the same set of depended-upon packages.

We believe cohesion is affected by internal and external factors and it should be treated based on this concept for accurate assessments. On the other hand, the generalised term of "cohesion" should not be used if the approach only relies on one consideration, i.e., internal or external. Terms such as "Common Closure Cohesion" and "Common Reuse Cohesion" can be used to describe the approach that relies on one consideration, i.e., internal and external, respectively. It is worth saying that Martin [21] has established a theory behind the internal and external factors by presenting the three package cohesion principles already discussed in Section II. Moreover, Martin's cohesion principles have been used to distinguish between package cohesion types in our previous work, Albattah and Melton [23].

V. Conclusion and Future Work

In this paper, a preliminary research survey on package cohesion approaches is presented. The survey shows that there is a rich variety of package cohesion understanding, which has led to the production of different package cohesion metrics in which each of them is based on a specific view of cohesion. We believe that there are significant differences in these metrics. Thus, the metrics of these approaches measure different things. The example given in the paper shows different values of cohesion and motivates us to classify the presented approaches. A preliminary classification reveals the reason behind the diversity of package cohesion approaches that led to distinct results in the given example. Obviously, the scope of cohesion is the foundation for this classification. We conclude that cohesion is more than one part and the term of "cohesion" should not be used unless the internal and external considerations are taken into account. Otherwise, terms such as "Common Closure Cohesion" and "Common Reuse Cohesion" can be used to describe the approach that relies on one consideration, i.e., internal and external, respectively.

In future work, we plan to examine the role of package cohesion in predicting software maintainability and software reusability.

References

[1] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design." IEEE Transactions on Software Engineering, 20.6 (1994): 476-493.

[2] V. Basili, "Evolving and packaging reading technologies." Journal of Systems and Software 38.1 (1997): 3-12.

[3] L. Briand, J. Daly, and Jürgen Wüst, "A unified framework for cohesion measurement in object-oriented systems." Empirical Software Engineering 3.1 (1998): 65-117.

[4] L. Briand, S. Morasca, and V. Basili, "Measuring and assessing maintainability at the end of high level design." Conference on Software Maintenance Proceedings, 1993. CSM-93 (pp. 88-87), IEEE, 1993.

[5] B. Henderson-Sellers, L. Constantine, and I. Graham, "Coupling and cohesion (towards a valid metrics suite for object-oriented analysis and design)." Object Oriented Systems 3.3 (1996): 143-158.

[6] S. Orlov and A. Vishnyakov, "Metric Suite Selection Methods for Software Development of Logistics and Transport Systems." Proceedings of the 11th International Conference "Reliability and Statistics in Transportation and Communication" (RelStat'11), 19-22 October 2011, Riga, Lativia, p.301-310.

[7] J. Eder, G. Kappel, and M. Schrefl, "Coupling and cohesion in object-oriented systems." Technical Reprot, University of Klagenfurt, Austria (1994).

[8] Y. Lee, B. Liang, S. Wu, and F. Wang, "Measuring the coupling and cohesion of an object-oriented program based on information flow." In Proc. International Conference on Software Quality, Maribor, Slovenia, 1995, (pp. 81-90).

[9] G. Gui and P. Scott, "Coupling and cohesion measures for evaluation of component reusability." Proceedings of the 2006 International workshop on Mining software repositories, 2006, (pp. 18-21). ACM, 2006.

[10] M. Hitz, and B. Montazeri, "Measuring coupling and cohesion in object-oriented systems." Proceedings of the International Symposium on Applied Corporate Computing. Vol. 50. 1995.

[11] W. Li and S. Henry, "Maintenance metrics for the object oriented paradigm." Proceedings of First International Software Metrics Symposium, 1993, (pp. 52-60), IEEE, 1993.

[12] S. Chidamber and C. Kemerer, "Towards a metrics suite for object oriented design." Vol. 26. No. 11 , 1991, (pp. 197-211). ACM.

[13] J. Bieman and Byung-Kyoo Kang, "Cohesion and reuse in an object-oriented system." ACM SIGSOFT Software Engineering Notes. Vol. 20. No. SI. ACM, 1995.

[14] J. Bieman and Linda M. Ott, "Measuring functional cohesion." IEEE Transactions on Software Engineering, Vol. 20, No. 8, (1994): (pp 644-657).

[15] L. Etzkorn, S. Gholston, J. Fortune, C. Stein, D. Utley, P. Farrington, and G. Cox, "A comparison of cohesion metrics for object-oriented systems." Information and Software Technology Vol. 46, No. 10, (2004): (pp 677-687).

[16] H. Chae, Y. Kwon, and Doo-Hwan Bae, "A cohesion measure for object-oriented classes." Software-Practice and Experience, Vol. 30, No.12, (2000): (pp 1405-1432).

[17] L. Ott, , J. Bieman, B. Kang, and B. Mehra, "Developing measures of class cohesion for object-oriented software." In Proc. Annual Oregon Workshop on Software Merics (AOWSM'95), vol. 11. 1995.

[18] J. Bansiya, L. Etzkorn, C. Davis, and W. Li, "A class cohesion metric for object-oriented designs." Journal of Object-Oriented Programming, Vol. 11, No. 8, (1999): (pp 47-52).

[19] V. Misic, "Cohesion is structural, coherence is functional: Different views, different measures." Proceedings of the Seventh International Software Metrics Symposium, 2001, (pp. 135-144), METRICS. IEEE, 2001.

[20] B. Xu, Z. Chen, and J. Zhao, "Measuring cohesion of packages in Ada95." ACM SIGAda Ada Letters, Vol. 24, No.1, (2004): (pp 62-67).

 ISBN: 978-1-61208-438-1

[21] R. C. Martin, Agile software development: principles, patterns, and practices. Prentice Hall PTR, 2003.

[22] L. Ponisio and O. Nierstrasz, “Using contextual information to assess package cohesion”, Technical Report No. IAM-06-002, 2006, Institute of Applied Mathematics and Computer Sciences, University of Berne, 2006.

[23] W. Albattah and A. Melton, “Package cohesion classification”, in: Software Engineering and Service Science (ICSESS), 2014 5th IEEE International Conference on, IEEE, 2014, (pp. 1–8).

[24] T. Zhou, B. Xu, L. Shi, Y. Zhou, and L. Chen, "Measuring package cohesion based on context." IEEE International Workshop in Semantic Computing and Systems, 2008. WSCS'08, (pp. 127-132), IEEE, 2008.

[25] H. Abdeen, S. Ducasse, H. Sahraoui, and I. Alloui, "Automatic package coupling and cycle minimization." 16th Working Conference on Reverse Engineering, 2009, (pp. 103-112), WCRE'09. IEEE, 2009.

[26] T. Biggerstaff and A. Perlis, "Software reusability: vol. 1, concepts and models." (1989).

[27] L. Briand, S. Morasca, and V. Basili, "Property-based software engineering measurement." IEEE Transactions on Software Engineering, Vol.22, No.1, (1996): (pp 68-86).

[28] M. Bauer and M. Trifu, "Architecture-aware adaptive clustering of OO systems." Eighth European Conference on Software Maintenance and Reengineering Proceedings 2004, CSMR 2004, (pp. 3-14), IEEE, 2004.

[29] O. Seng, M. Bauer, M. Biehl, and G. Pache, "Search-based improvement of subsystem decompositions." In Proceedings of the 7th annual conference on Genetic and evolutionary computation, 2005, (pp. 1045-1051), ACM, 2005.

[30] N. Tagoug, "Object-oriented system decomposition quality.", 7th IEEE International Symposium on High Assurance Systems Engineering Proceedings, 2002, (pp. 230-235), IEEE, 2002.

 ISBN: 978-1-61208-438-1