Towards A Design Measurement Context for Software Coupling and Cohesion Requirements

KHALED ALMAKADMEH, KHALID T. AL-SARAYREH
Department of Software Engineering
Hashemite University
Zarqa 13115, Jordan
khaled.almakadmeh@hu.edu.jo, khalidt@hu.edu.jo

KENZA MERIDJI
Department of Software Engineering
Petra University
Amman 11196, Jordan
kmeridji@uop.edu.jo

Abstract— Cohesion and coupling are often mentioned together; this can be explained as they represent or share similar concepts. The review of literature shows that a key element for both cohesion and coupling is the number of interactions between components. This paper proposes a new measurement method that accommodates different views of the most important design concepts of cohesion and coupling and setup typical scenarios to illustrate how to use the method concepts with practical software artifacts.

Keywords:Coupling, Cohesion, Software Measurement, ISO1971, COSMIC

1. Introduction

The concept of cohesion is frequently linked with coupling. Even if they could be interpreted as separate concepts, one often needs to measure coupling in order to evaluate cohesion. Indeed, this problem is pointed out by Lethbridge and Anquetil [1]. Counsell et al. says, any measure of cohesion which uses parameters of class methods, the attributes cannot avoid including a high degree of coupling to other classes [2]. They also add that Comprehension of class cohesion is largely an exercise in comprehension of class coupling [2].

Furthermore, cohesion can be seen from two main perspectives. First, cohesion can be generally defined as how the elements making up a module are related [3]. Another interpretation considers the functional point of view, which is a crisp abstraction of a concept or feature from the problem domain [4]. In [4], different methods used to express cohesion: structural and semantic metrics, information theory-based, slice-based, and knowledge-based & approaches using data mining. The most common type is the structural metrics. The idea behind this paradigm is that class variables are referred and shared between methods, which in turn influences the degree of cohesion. It is mainly based on the relationships between methods of a class. They argue that all structural metrics capture the same aspects of cohesion data flow between the methods of a class [4].

We did a review of some papers discussing the measurement of cohesion and/or coupling. The main objective was to find similar key concepts for cohesion or coupling. It is said that software design principles are key notions considered fundamental to many different software design [3]. Most notably, the measures of coupling and cohesion are part of these and are also recognized as key concepts of design. In order to measure a software design, we will focus on these two major design concepts.

While it is recognized that high cohesion and low coupling can lead to a good software design, the separation and redefinition of these concepts or related terms could change how we perceive design. Accordingly, no assumption will be made about good or bad values for these concepts. In fact, that is more the role of the interpretation phase, which can analyze the output values of the measurement methods and suggest adjustments.

Facts for both cohesion and coupling:

- The measure of cohesion seems to be dependent of coupling [1, 2].

- A collaboration of objects may include one type of class or different classes participating together [5].
- Some basic properties have been proposed for cohesion and coupling measures [6].

This paper is divided into five sections; section 2 presents the review of the literature; section 3 presents the measurement objectives; section 4 presents the characterization of the mesurand. Finally, section 5 concludes the paper findings.

2. Related Work

Briand et al. [7] proposed four cohesion properties that a valid measure should support, arguing that a measure must be supported by some underlying theory. The four cohesion properties defined are one of the more recent proposals to characterize cohesion in a reasonably intuitive and rigorous manner [8].

These four properties are: non-negativity and normalization (greater than 0 and less than a fixed value), null value and maximum value, monotonicity and merging of unconnected classes [9].

Concerning the coupling measure, Briand et al. [7] also defined the properties that should be validated; such measure should be nonnegative and null when no relationships between modules exist. Further, when new relationships are created, coupling should not decrease. On the other hand, when modules are merged, it can only decrease the coupling, since inter-module relationships may have been removed [6]. The goal of these properties is to allow better comparison of the measurement values for cohesion and coupling.

Marcus et al. [4] proposed a way to measure the model type of cohesion; a class that represents a single, semantically meaningful concept. To measure type of cohesion, they suggest that the responsibilities associated with the classes have to be recorded in the code through identifiers and comments. By analyzing the semantic information in the code, they can find the measure of cohesion. Authors in [4] defined a conceptual similarity between methods and the cohesion as the average of all values of conceptual similarity in the methods of the class.

Stevens, Myers and Constantine [10] defined an association-based cohesion on an ordinal scale and categorized several types of cohesion. Byung-Kyoo and Bieman [11] used these definitions as a base for a measure of design and code cohesion.

In order to measure the cohesion of a design, Byung-Kyoo and Bieman [11] models the data, the control dependencies relationships between input and output by an input-output dependence graph (IODP). Data dependence is defined if there is a “definition-use” or “use-definition” relation. A variable has a control dependency on another if the value of the latter determines if the first statement will be performed or not. There are other special types of dependences defined. In this case, the measurement method is called the DLC (design level cohesion). The cohesion level of the module is the weakest (lowest) of all the pairs of methods.

Byung-Kyoo and Bieman [11] also uses functional cohesion and defines three measures based on data slices. The data slice of a variable is the sequence of data tokens which have a dependence relationship with that variable. Moreover, glue tokens are data tokens common to more than one data slice, while superglue tokens are common to every data slice of a module. Using that technique, Weak Functional Cohesion (WFC) can be expressed as the number of glue tokens divided by the total number of tokens in a method. The Strong Functional Cohesion (SFC) is the ratio of superglue tokens on the total number of data tokens in a method. Finally, authors in [11] proposed another measure, which is called Design-Level Functional Cohesion (DFC); for such measure, IODG includes dependency relationships between input and output components. The cohesiveness of the i^{th} component of a module is defined as the number of outputs in a dependency relation with the i^{th} component divided by the number of outputs in the module’s IODG. According to this study [11], the measures explored represent different aspects of cohesion. Namely, the connections, the isolated components and the essential components (connected with all methods outputs).

The majority of existing cohesion metrics capture class cohesion in terms of connections exists among members within a class [5]. To address this problem, Badri et al. [5] introduced a new parameter Common Objects Parameters which is described as methods having a same object as parameter. They argue that methods can be functionally related even if they do not share any instance variables. Two types of collaboration levels are then presented; the first collaboration level implies that several objects, belonging to different classes, participate to realize the functionality. The other collaboration level refers to a collaboration of methods within the same class, using objects as instance variables or by passing arguments.

Makela et al. [12] criticize the widely-spread LCOM lack of cohesion metric and its variations. They propose to consider an external view of cohesion, which is defined as how other classes use the features of another class [12]. Since the client classes often use only a subset of all methods, some methods should be excluded in the cohesion measure.

Authors in [12] explain that LCOM value is affected by these kinds of methods that do almost nothing. For example, if a class contains a lot of getters and setters methods, the LCOM metric will give an erroneous bad cohesion. In this case, the proposed metric ELCOM would have a low internal cohesion but a high external cohesion. This is because clients tend to use the same key methods, which would use all the instance variables of the class; ELCOM is higher if the number of instance variables used by client classes is greater. Also, LCOM tends to be affected by the size of the class. Counsell et al. [2] has shown that the size, as defined by the number of methods of a class, is a confounding factor. Counter-intuitively, small classes are not necessary more cohesive than larger ones. The idea is to classify methods as special or normal. Special methods are access (only reads from or write to an attribute), delegation (delegate a message to another object).

Authors in [12] also include constructors and destructors which initialize or de-initialize essential attributes of the class. Since it has been demonstrated that these special methods do not influence cohesion, they need to be excluded from the calculation. A class member dependence graph is built and this abstract model is composed of four types of relationships (represented as edges): read, write, call and flow dependencies among these nodes. A node can be of two types: a normal method or an attribute. The dependence degree is then expressed in terms of the number of attributes or methods that the node depends on. The cohesion of the class is the average of the dependency degrees of all normal methods and attributes.

Misic et al. [13] introduce the notion of coherence, which is similar to the functional cohesion. It is defined as how well its members contribute towards a common, externally defined purpose or objective [13]. This measure is one of the few measures that can actually be applied early in the software development cycle and is focused on the customer. The authors in [13] argue that cohesion is to be measured against the yardstick of the objective, purpose, or function of a software module this purpose is to be found outside the module. Therefore, cohesion has to be an external property. They also suggest that other measures such as coupling could be used to assess the internal organization.

The read and write range and scope definitions are then explained. An object can potentially be written by other objects (write scope), but only some of them really write to that object (write range or suppliers). Similarly, the set of all objects that can potentially read (or use) another object is the read scope. The set of objects that actually read (or use) another object is the read range (also called clients).

At this point, it is possible to express the coherence of one client object as the ratio between the number of objects writing that client object and the number of objects in the boundary considered. This ratio can also be seen as the proportion of objects in the boundary set that are writing to the client object. The coherence of the boundary as a whole is the average of these ratios, for all clients. Coherence has a value of zero when all clients use only one object of a set S and has the maximum value when each client uses all objects of S. In addition, it is possible to focus on either external or internal coherence, by restricting the reference set S accordingly.

Counsell et al. [2] defines the HD metric which uses the Hamming Distance to measure cohesion. They build a matrix of (m) × (p), where m is the number of methods and p is the number of different parameters in all methods of the class. The cell value of the matrix, assigned for each method, takes a value of one if the method uses the parameter type in its parameter list. Otherwise, it is set to zero.

3. Measurement Objectives

Cohesion and coupling are often mentioned together; maybe because they represent or share similar concepts. In fact, this is reflected in the literature. While several authors consider internal and external views for cohesion [12, 13, and 14], the distinctions between coupling and external cohesion is less clear. In addition, in order to evaluate the cohesion of a component, one often needs to measure coupling [1, 2].

Any design supports some quality requirements. For example, one design can be easier to adapt to changes than another. In the current context, the study of quality attributes is separated from requirements of design. While a complete design supports quality properties, measurement methods for design and quality in general are different. For the rest of this paper, the focus will be only on the measurement of the quality of a design.

In this context, the objective of the measurement is to propose a measurement method for cohesion. Whenever possible, the synthesis of the literature summarized in section 4 will guide the design of the measurement method, while taking several constraints into account for the current work. For instance, the measure should help to estimate design requirements early in the software development lifecycle. Also, the design of the measure should be customer-centric and not from the developer point of view. Few measures are valid early development cycle and focus on the customer point of view. In fact, most measures proposed in the literature try to analyze the code of software [15] that takes place in the implementation phase and mostly concerns the developers. It can be assumed that a good internal design will affect positively the end users of the software. the main objective will be seeking an enlarged consensus and this will remain the priority in case of conflicts with other constraints. Whenever possible, the generality of the proposed model will be preserved.

Cohesion is not the only criterion to evaluate the quality of a design but it is a major one. Coupling is also a critical property of a software design but it will not be considered for the scope of this paper. However, since cohesion and coupling share some concepts, additional information is given informally to guide future research.

4. Characterize the Measured

According to the synthesis of a few papers concerning cohesion, presented in section 2, some key concepts appear frequently and can help to define the concept of cohesion. Every measure of cohesion considers the interactions between a class and its attributes or methods. The concept of collaboration between objects is also present. In addition, internal and external views of cohesion are often mentioned.

4.1 Definitions

The terms on which the meta-model will be built are defined in this subsection.
Software Design: the process of defining architecture, components, interfaces and other characteristics of a system or component and the result of that process [16] a software design (the result) must describe how software is **decomposed** and organized into **components** and the **interfaces** between those components [3].
Component: one of the parts that make up a system. A component may be hardware or software and may be subdivided into other components [16]. Note: the terms “module,” “component,” and “unit” are often used interchangeably or defined to be sub-elements of one another in different ways depending upon the context. The relationship of these terms is not yet standardized [16].
Interface: “hardware or software component that connects two or more other components for the purpose of passing information from one to the other [16].
Message: information exchanged on an interface.
Attribute: characteristic of an item; for example, the item's color, size, or type [16].
Cohesion: how the elements making up a module are related [3].
Coupling: strength of relationships between modules [3].

4.2 Meta-Model

The decomposition of any system creates components and subcomponents. Layers, modules, classes and functions (or methods) are examples of components in software. However, when defined at a higher level of abstraction, a component becomes more of a boundary than a concrete component. For this reason, some components may exhibits properties but the boundaries do not have attributes, for they exist only to regroup other components.

The functionality of the software is distributed among the components making up the system. Figure 1 below illustrates the main concepts for the measurement of the cohesion for any component.

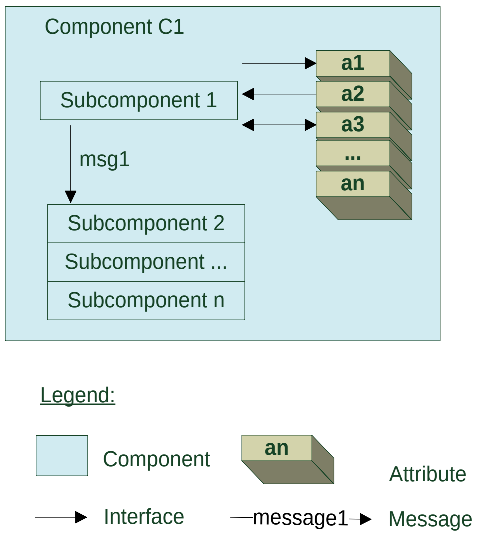

Figure 1 A component and its attributes

Any component can support any number of attributes or characteristics. In addition, a component can have no attributes, depending on the context. In a cohesive component, the interactions between the component, its attributes and subcomponents are conceptually related. A greater number of interactions between a component and its elements contribute to enhance the cohesion of the component. Therefore, the mesurand for the cohesion of a component is the number of interactions between the component, its attributes and subcomponents.

As of now, it is not clear if the cohesion depends on the distribution of the interactions or the number of interactions. We consider the number of interactions. In that case, an attribute could be used proportionally more than the others, and the component could still be cohesive. However, when considering the cohesion of a whole collaboration of objects, as it will be seen later, the number of interactions seems to become more relevant than its distribution.

In order to evaluate the number of interactions, the concept of data groups, used in COSMIC [17] is used. For instance, the attributes of a component form a data group. Cohesion of a component can be expressed as the number of data movements between its attributes and subcomponents.

4.3 Numerical Assignement Rules

The interactions within a component describe its internal data movements. A first set of data movements is the interactions between the component and its attributes. For instance, a component can read or change (write) one or several of its attributes. The second set of data movements represents interactions between the component and its subcomponents. A component can use some of its subcomponents to realize the functionality of the software.

$$\begin{aligned}&\text{Internal component interactions (CFP)}\\&= \text{data movements between its attributes (CFP)}\\&+ \text{ data movements between its subcomponents (CFP)}\end{aligned}$$

In addition, the subcomponents interactions need to be taken into account. Thus, the total interactions within a component are the internal interactions added to the interactions occurring inside all of the subcomponents. Since a subcomponent is also a component; such definitions is applied recursively.

$$\begin{aligned}&\text{Component interactions (CFP)}\\&= \text{Internal component interactions (CFP)}\\&+ \text{Subcomponents interactions (CFP)}\end{aligned}$$

The subcomponents can be classified as related or unrelated subcomponents; the related subcomponents are those participating in the internal component interactions. For instance, if a subcomponent uses or depends on an attribute or another subcomponent, that subcomponent is related to its parent component. Otherwise, the subcomponent is said to be unrelated, which means the subcomponent is independent of the others.

Once subcomponents have been classified, it is possible to evaluate the component interactions and they can be added together. When evaluating the interactions of the related components, only the subset of related subcomponents is considered. All of the subcomponents interactions can also be counted, whether they are related or not.

$$\begin{aligned}&\text{Interactions of a set of components (CFP)}\\&= \sum \text{Component interactions (CFP)}\end{aligned}$$

$$\text{Cohesion ratio} = \frac{\text{Internal component interactions (CFP)} + \text{Related subcomponents interactions (CFP)}}{\text{Internal component interactions (CFP)} + \text{All subcomponents interactions (CFP)}}$$

The measure of a cohesion is then on a ratio scale and can take any value between 0 and 1. The cohesion ratio of a component can be seen as the proportion of its related functionality.

If a component has no interactions between its attributes and subcomponents, then the cohesion ratio is 0. The cohesion ratio is undefined if there are no subcomponents and no interactions between its attributes. In that case, the cohesion cannot be evaluated. When a component has no subcomponents, the cohesion ratio is 1 since the component forms a self-contained entity that is entirely independent. Moreover, if all the interactions between components are related, the cohesion ratio raises up to 1.

The components should be located within the same layer, since different layers can involve different kinds of technologies, and it would be harder to compare the values of the measurements. All measurements must be done using the same level of granularity in order to be comparable. In addition, the same procedure rules needs to be applied to compare cohesion using similar criteria. Depending on the level of details of the specifications documents, more precise interactions can be captured between components of the system.

5. Conclusion

The current work tried to find the key design concepts were and how they were defined in the literature. Two key design concepts were identified: cohesion and coupling. The review of literature shows that a key element for both cohesion and coupling is the number of interactions between components. Also, it has been found that the measure of cohesion mostly depends on coupling.

A measurement method for cohesion was proposed. Some typical scenarios were presented to better illustrate how to use the meta-model with practical software artifacts. For instance, the methodology to measure the cohesion of several types of components was described.

There is a probability that a collaboration of objects refer to similar concepts, since the objects work together to accomplish a related functionality. Hence, the measure of the cohesion of a collaboration of objects could be appropriate in object-oriented systems. Further study could try to experiment with the proposed measurement method and adjust it. Also, the same process could be done for coupling, in the hope to reveal more clear differences between the key concepts in a software design.

6. References

[1] T. Lethbridge, N. Anquetil. *Experiments with Coupling and Cohesion Metrics*. Online-resource: www.site.uottawa.ca/~tcl/papers/metrics/ExpWithCouplingCohesion.html

[2] S. Counsell, E. Mendes, S. Swift, *Comprehension of object-oriented software cohesion: the empirical quagmire*. Proceedings of the 10th International Workshop on Program Comprehension, Paris, France, 2002, pp.33-42.

[3] P. Bourque and R.E. Fairley, *Guide to the Software Engineering Body of Knowledge, Version 3.0*, IEEE Computer Society Press, 2014.

[4] A. Marcus, D. Poshyvanyk, *The conceptual cohesion of classes*. Proceedings of the 21st IEEE International Conference on Software Maintenance – ICSM'05, Budapest, Hungary, 2005, pp. 133-142.

[5] L. Badri, M. Badri and G. A. Badara, “Revisiting Class Cohesion: An Empirical Investigation on Several Systems”, Journal of Object Technology, Vol. 7, No. 6, 2008, pp. 55-75.

[6] L. C. Briand, J. Daly, V. Porter et J. Wust, *A comprehensive empirical validation of design measures for object-oriented systems*. Proceedings of the 5th International Software Metrics Symposium, Bethesda, Maryland, 1998, pp. 43-53.

[7] L. C Briand, S. Morasca et V. R. Basili, *Property-based software engineering measurement*. IEEE Transactions on Software Engineering, Vol. 22, No. 1, 1996, pp. 68-86.

[8] Z. Yuming, B. Xu, J. Zhao and H. Yang. *ICBMC: An Improved Cohesion Measure for Classes*. Proceedings of the International Conference on Software Maintenance. Monteal, Canada, 2002, pp. 44-53.

[9] C. Zhenqiang, Y. Zhou, B. Xu, J. Zhao and H. Yang, *A Novel Approach to Measuring Class Cohesion based Dependence Analysis*. Proceedings of International Conference on Software Maintenance, Montreal, Canada, 2002, pp. 377-384.

[10] W. Stevens, G. Myers and L. Constantine, Structured design, *IBM Systems Journal*, No. 2, 1974, pp.115 -139.

[11] K. Byung-Kyoo and J. M. Bieman, *Design-level cohesion measures: derivation, comparison, and applications*. Proceedings of 20th International Computer Software and Applications Conference, Seoul, Korea, 1996, pp. 92-97.

[12] S. Makela, V. Leppanen, *Client based Object-Oriented Cohesion Metrics*, 31st Annual International Computer Software and Applications Conference, Beijing, China, 2007, pp. 743-748.

[13] V. B. Misic, *Cohesion is structural, coherence is functional: different views, different measures*. Proceedings of the 7th International Software Metrics Symposium, London, England, 2001, pp. 135-144.

[14] T. Zhou, B. Xu, L. Shi, Y. Zhou and L. Chen, *Measuring Package Cohesion Based on Context*. IEEE International Workshop on Semantic Computing and Systems, Huangshan, China, 2008, pp. 127-132.

[15] L. Stein, S. Gholston, P. Farrington and J. Fortune. 2006. *A Knowledge-Based Cohesion Metric for Object-Oriented Software*. Online-resource:http://www.dcc.ufla.br/infocomp/artigos/v5.4/art06.pdf

[16] Institute of Electrical & Electronics Engineers, IEEE Standard Glossary of Software Engineering Terminology. IEEE Std 610.12-1990.

[17] International Organization for Standardization, ISO19761: A Functional Size Measurement Method – COSMIC, Geneva, Switzerland, 2011.