Journal of Theoretical and Applied Information Technology
30th April 2018. Vol.96. No 8

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

SYSTEM COUPLING AND COHESION REQUIREMENTS MODEL (SC^{2}RM): MEASUREMENT APPROACH FOR REAL TIME SYSTEM

KHALED ALMAKADMEH

Department of Software Engineering, the Hashemite University, P.O. Box 330136, Zarqa (13115) Jordan

E-mail: khaled.almakadmeh@hu.edu.jo

ABSTRACT

One of the main challenges for software development organizations is to build software systems with measured complexity. Monitoring a software system complexity help software engineers in development phases of system development life cycle, such as software system reusability and software system maintainability. A key measure of software complexity is the degree of cohesion and coupling within and between its components. The literature emphasizes that a key system element to measure the degree of cohesion and coupling is the number of interactions between software components. This paper propose a new model to measure the degree of cohesion and coupling within and between real-time system components based on ISO19761 international standard. A case study is conducted to verify the applicability of the proposed measurement model using structural specifications and First Class Relation. The resulting measures are valuable indicators of a software system complexity that directly affects its reusability and maintainability.

Keywords: *Coupling; Cohesion; Software Measurement; Real-time System, ISO19761.*

1. INTRODUCTION

Software development organizations face a disturbing fact that the cost to maintain software systems is typically much higher than the cost of development for these systems [1]. One of the main causes for this high cost is the high complexity within and between system components; which makes it more difficult to change and/or upgrade the functionality of these system components [2]. In other words, when the requirements of such systems change in a continuous basis, it is then compulsory to change system components to accommodate such new emerging system requirements.

Several research studies are proposed to measure the degree of complexity in software systems under development and even for maintenance system projects [3-6]. On the other hand, these research studies have measured the degree of complexity using system's specifications at a late phase of the system development life cycle. For software system under development, it is important to measure the degree of system complexity at an early phase of the system development life cycle, in order to use such complexity measures to build reliable effort estimation models.

A key measure for complexity of system components is the degree/level of cohesion and coupling exist within and between system components. Several research studies in the literature have had used the number of interactions within and between systems' components as a measure of cohesion and coupling exist in these components [7].

Lethbridge and Anquetil [4] have reported that software engineers needs to measure coupling in order to measure the degree of cohesion. Counsell et al. [5] emphasized that any measure of cohesion that uses parameters of class methods or attributes cannot avoid including a high degree of coupling to other classes. They reported that comprehension of class cohesion is an exercise in comprehension of class coupling.

Cohesion can be seen from two main perspectives. First perspective defines cohesion as how related the elements that making up a system module [6]. Second perspective, considers a functional point of view, which is a crisp abstraction of a concept or feature from the problem domain.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Marcus and Poshyvanyk [7] have experimented several methods to express cohesion, such as structural and semantic metrics [7], theory-based metrics [8], and slice-based metrics [9]. They reported that most commonly used metrics are structural metrics. For example, class variables are referred and are shared between different class methods, and this reflects the degree of cohesion. Further, this study reported that all structural metrics capture the same aspects of cohesion data flow between the methods of a class.

Stevens et al. [10] have defined an association-based cohesion on an ordinal scale and categorized several types of cohesion. Table 1 below presents six different types of cohesion as defined in [10].

Table 1: Different types of cohesion [10]

Type of Cohesion	Description
Coincidental relation	Common input of two modules has no dependence relationship, and neither their output.
Conditional relation	Output of two modules is control dependent on a common input, or an output holds c-control dependence and another has *i*-control dependence on the input.
Iterative relation	Output of two modules is control dependent on a common input.
Communicational relation	Output of two modules based on common input: first output holds data dependence and second holds either a control or a data dependence.
Sequential relation	Output is dependent on other output.
Functional relation	One output in a module is only exist.

Facts for both cohesion and coupling:

- A measure of cohesion is dependent on coupling [3, 4].
- Collaboration of objects may include one type of class or different classes participating together [11].

The motivation of this research paper is to help software development organizations and in particular software project managers and technical leaders to build more accurate effort estimation models, by improving one of the inputs (i.e. measurement of cohesion and coupling) for the effort estimation process. This improvement will improve planning, management, and development of software at different phases of the software development life cycle. Further, the measurement results of the proposed model can be used for software benchmarking purposes conducted by specialized groups such as the International Software Benchmarking Group (ISBSG).

The contribution of this paper is a new measurement model to measure the degree of cohesion and coupling exist within and between system components based on ISO19761 international standard. This measurement model measures functional size of interactions exist within and between system components independently from development technology used to develop the software system.

This paper is organized as follows: section 2 present the literature review and section 3 presents the design of the measurement model based on international ISO standard. Section 4 presents a verification of the applicability for the proposed measurement model using First Class Relation. Finally, section 5 presents conclusions and future work directions.

2. LITERATURE REVIEW

Badri et al. [11] proposed an attribute called "common object parameter" which is defined when different class methods have the same attribute. They consider that class methods are related functionally, even if they do not share instance variables. Badri et al. [11] have defined two collaboration levels, first collaboration level is defined when several objects that belong to different classes participate to achieve certain functionality. Whereas, second collaboration level is defined when different methods within the same class collaborate using objects, such instance variables or passing arguments.

Briand et al. [12] proposed four cohesion properties that a valid measure should have, arguing that a cohesion measure should be supported by some underlying theory. The proposed properties are non-negativity, normalization (greater than 0 and less than a fixed value), null value and maximum value, monotonicity and merging of unconnected classes [13].

Marcus et al. [14] proposed an approach to measure model type of cohesion, which represent a single, semantically meaningful concept. They suggested to document responsibilities associated with classes in code using identifiers and comments. Then, analyze semantic information in

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

code to measure the level of cohesion. Marcus et al. [14] defined conceptual similarity between methods and cohesion as the average of all values of conceptual similarity in the methods of a class.

Byung-Kyoo and Bieman [15] used the concepts proposed in [10] as a base to measure design and code cohesion. They proposed to model data and control relationship dependencies using an input-output dependency graph. A data dependency is defined if there exist a 'use-definition' relationship. On the other hand, an attribute has a control dependency on another if the value of the latter determines if the first statement is performed or not. Further, special types of dependencies are defined; a design level cohesion is defined as the lowest level of all pairs of methods.

Byung-Kyoo and Bieman [15] have defined three measures of functional cohesion based on data slices. A data slice of an attribute is the sequence of data tokens that have a dependency relationship with that attribute. Further, glue-tokens are data tokens that are common to more than one data slice, while super-glue tokens are data tokens that are common to every data slice in a module. Based on these concepts, a weak functional cohesion (WFC) is expressed as the number of glue tokens divided by the total number of tokens in a method. Whereas, strong functional cohesion (SFC) is defined as the ratio of superglue tokens on total number of data tokens in a method.

Makela et al. [16] conducted a study to evaluate an LCOM (lack of cohesion) metric. They defined an external view of cohesion as how a specific class use the features (i.e. methods) of another class. A client class typically use a subset of methods, and therefore methods not used by the client class are excluded in the measure of cohesion. Further, Makela et al. [16] have included constructors and destructors that initialize or de-initialize essential attributes of a class in the measurement of cohesion. A class-member dependence graph is built to represent four types of dependency relationships as edges: read, write, call and flow dependencies among nodes. A node in this graph represent an attribute or a class method and the degree of cohesion is measured as the average dependency degrees of all attributes and methods.

Soares et al. [17] proposed two UML profiles to enable software engineers to produce less coupled system components. However, the two proposed profiles are not experimented using a case study to verify their applicability. Agner et al. [18] applied black box testing in a model driven architecture context aimed to produce more coherent model transformations.

Madhwaraj [19] has conducted an empirical study to compare two metrics used to predict maintainability of packages in object-oriented systems. Coupling of packages is calculated using both metrics as a primary input to measure the degree of maintainability.

Újházi et al. [20] proposed two metrics to measure the degree of coupling and cohesion of object classes in a large open-source software system, using the concepts of coupling metric (CCBO) and conceptual lack of cohesion on methods.

Rajkumar et al. [21] proposed a set of equations to measure the degree of coupling and cohesion using object-oriented Java code. Further, Maheshwari et al. [22] proposed a coupling metric for Java classes and they did not consider the calculation of coupling at higher levels such as package level.

Ludwig et al. [23] proposed an open-source plug-in to measure the architectural complexity of software as an indicator of software product maintainability. The proposed plug-in calculate the architectural complexity using complexity metrics such as Lines of Code (LOC), Weighted Method Count (WMC) and Response for Class (RFC).

Almugrin and Melton [24] conducted an experimental study to validate three package metrics (i.e. coupling, stability and abstractness) built based on Martin metrics [25] for software package responsibility to produce an early indicator to software maintainability and testability. The experimental study conducted using three open-source software projects to investigate software package responsibility based on direct dependency and the experimental results yield an improved prediction of software maintainability and testability.

Faragó et al. [26] conducted an experimental study to investigate the impact of three (3) version control metrics such as intensity of modifications, code ownership and aging on software maintainability using fourteen (14) versions of four (4) open-source software projects. The experimental results showed a high correlation between version control metrics and corresponding maintainability indicators such as post-release defects.

Shafiabady et al. [27] presented a summary of prediction models for software maintainability: these prediction models employ different

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

information from a software product including modularity, testability, modifiability, size and structural complexity of UML class diagrams…etc.

Gonzalez et al. [28] conducted an empirical study that applies automated unit testing frameworks to investigate the benefit of using xUnit testing patterns to improve the quality of maintainability attributes such as ease of diagnoses, modifiability, and comprehension. This study investigated more than eighty-two thousands open source projects. The results reported that only twenty-four percent of the investigated projects had test files implemented patterns that could help in software project maintainability. Further, the study reported that the decision to implement test patterns depend on developer decision rather than project characteristics.

Jain et al. [29] proposed a genetic algorithm to predict maintainability of two versions of four open source software. The number of changes counted in the source line of code of software to another in order to calculate the maintenance effort. Prediction models built using machine learning classifiers and then analyzed mean absolute error (MAE) and root mean square error (RMSE). The analysis results compared with other machine learning techniques such decision table, radial basis function neural network, and Bayes-Net and sequential minimal optimization. The analysis results showed that prediction models built using genetic algorithms produce improved prediction than typical machine learning techniques.

Rongviriyapanish et al. [30] proposed a prediction model to assess java class changeability. The prediction model built using the multilayer perceptron classifier (MLP) on 137 java classes from an open-source software project. According to the multilayer perceptron classifier, the proposed prediction level was able to predict the changeability of java classes on only one level rather than three levels.

Mo et al. [31] proposed an architecture maintainability metric called Decoupling Level (DL) derived from Baldwin and Clark's option theory. The proposed metric aimed to measure the degree on which software product designed as small and independent set of replaceable modules. The proposed metric experimented using multiple releases of 108 open source projects and 21 industrial projects.

Panca et al. [32] conducted a study aimed to implement a design pattern combination that develop maintainable mobile application services. The design pattern combination is singleton, memento, state, iterator, factory, builder, and flyweight. The combination experimented using three mobile applications from three different domains. The experimental results showed that design patterns such as singleton, memento, and iterator degrade modularity of the three mobile applications. Further, the design patterns factory and builder can improve and/or reduce modularity depending on the mobile application itself.

Baqais et al. [33] conducted an empirical study to analyze the relationship between class stability and software maintainability. A correlation presented in this study between both concepts using class stability metric proposed by [34] that measure class stability based on eight class properties and maintainability index used to measure the degree of maintainability from source code.

3. DESIGN OF A MEASUREMENT MODEL FOR COHESION & COUPLING

This section present the design of a measurement model to measure the degree of cohesion and coupling exist within and between software systems components based on ISO19761 international standard. Four steps are recommended by Abran [35] to design a reference measurement model as follows:

1. Determination of measurement objectives.
2. Characterization of cohesion and coupling terms.
3. Construction of cohesion and coupling metamodel.
4. Identification of numerical assignment rules.

3.1 Determination of Measurement Objectives

This part presents the main objective of the proposed measurement model for cohesion and coupling of software system components as part of the assessment process of software system complexity, along with the anticipated uses of the measurement results:

- Measurement objective: to measure the functional size of cohesion and coupling within and between software system components using ISO19761: COSMIC as an intentional standard for software functional measurement recognized by ISO.
- Intended use of measurement results: the uses of measurement results of cohesion and coupling of system components span the whole software development life cycle. These functional size measures represent one of the primary inputs for

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

the effort estimation process of software systems.

3.2 Characterization of Cohesion and Coupling Terms

This part presents characterization of terms (i.e. vocabulary) as defined in the IEEE-24765 standard of systems and software engineering vocabulary [36]. It is worth mentioning that the terms exist in the IEEE-24765 [36] standard are aligned with the terms and/or concepts defined by ISO standards such as ISO-41413-1 [37]. There are key concepts that 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. For the purpose of this research, the following terms are used to help in measurement of cohesion and coupling of software systems:

- Software design: is the process of defining architecture, components, interfaces and other characteristics of a component or a system. The result of the design process must describe how software is decomposed and is organized into components and interfaces between such components [6].
- Component: is one of the parts that make up a software system. A component may be hardware or software and subdivided into other components [19]. Note: the terms "module" "component" and "unit" often used interchangeably or defined to be sub-elements of one another in different ways depending upon the context.
- Interface: is "hardware or software component that connects two or more other components for the purpose of passing information from each other [19]. An interface can be classified into the following types:
  - Interface components: components that allow high-level interaction between interface functions.
  - Interface specifications: specifications that describe level of interaction required for interface component functions.
- Message: are information exchanged on an interface. Messages have two levels: functional and services levels. Messages consist of three types of data architecture movements as follows:
  - Messages exchange at functional level.
  - Message of intermediary services at system level.
  - Data exchanges between system components: direct exchange of data movements and indirect exchange of data movements.
- Attribute: is a characteristic of an item [36].
- Cohesion: is defined as how the elements that make up a module are related [6].
- Coupling: is defined as the strength of relationships between modules [6]. Coupling is the manner and degree of interdependence between software modules; a measure of how closely two routines or modules are connected is the strength of relationships between modules. Coupling refers to interdependencies between modules, while cohesion describes how functions within a module are related.

Figure 1 presents an example of coupling and cohesion relationship over its components in a software system. In this example, three modules are interconnected with each other in manner that allow them to accomplish their functionality. Cohesion within a certain software module is represented using bold-connected line. Whereas, coupling between software modules is represented using bold-dotted lines.

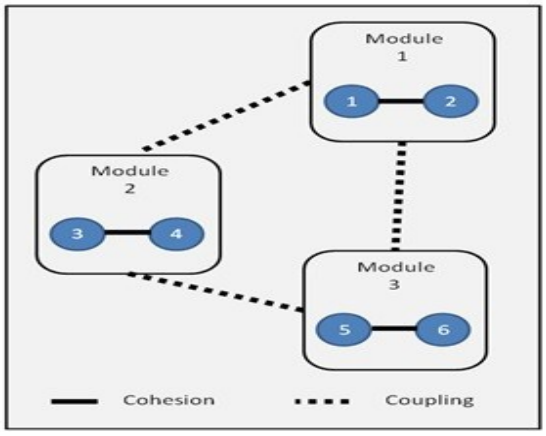

Fig. 1 A generic view of cohesion and coupling within and between software modules

3.3 Construction of Cohesion and Coupling Metamodel

The decomposition of a software system normally yields components and sub-components: layers, modules, classes and functions (or methods) are examples of components in the software engineering domain [36]. On the other hand, when such components and sub-components are defined at a higher level of abstraction; a component

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

becomes more of a boundary than a concrete component. Certain software components may exhibits properties but boundaries do not have attributes (i.e. properties), but they are exist only to regroup other software components. Normally, the functionality of a software system is distributed among different components making up such system.

Figure 2 presents an instance metamodel that represent the main concepts that are required for the measurement of cohesion in a software component. In this figure, a software system consists of software layers, certain layer typically consists of one or more software components, and such software components consist of one or more software sub-components. Normally, software sub-components exchange messages between each other within the same software component or with other software sub-components in different software component. A sub-component shall access certain attributes using a software/hardware interface in order to accomplish its functionality.

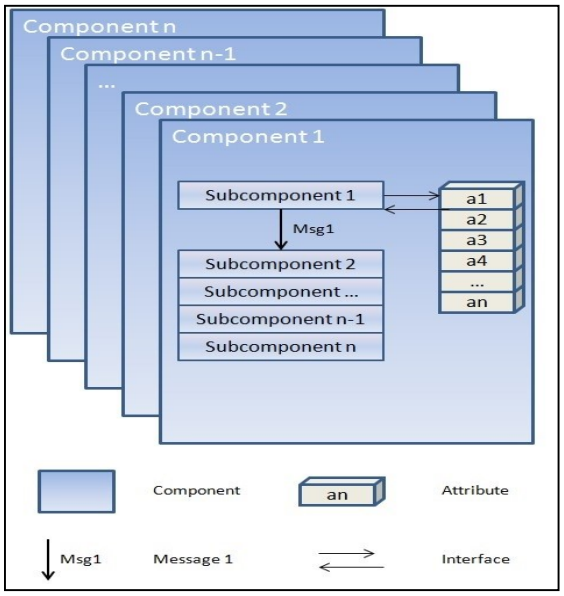

Fig 2 An instance metamodel for a sample software layer and its attributes

It is worth mentioning that – to date – it is still not standardized that the measure of the degree of cohesion depends on the distribution of the interactions or the number of interactions within a certain software component. For the purpose of this research, the number of interactions within a software component is adopted as a measure the degree of cohesion in that software component. Therefore, an attribute can be used proportionally more than other attributes and the whole software component is still cohesive. On the other hand, when considering cohesion for a whole set of collaborating objects, the number of interactions becomes more relevant than its distribution as a measure of the degree of cohesion. To count the number of interactions in a standardized way, the concept of data groups that is defined in the international standard for software functional size measurement ISO19761 [38] is adopted. For example, attributes of a software component represent one data group, and therefore, cohesion in a software component is measured as number of data movements between its attributes and its subcomponents.

3.4 Identification of Numerical Assignment Rules

Numerical assignments rules can be described using a descriptive text (i.e. a practitioner's description) or using mathematical expressions (i.e. formal theoretical viewpoint). According to the international standard for software functional size measurement – ISO19761 [38], a functional process is defined as an elementary component of a set of functional user requirements. It includes a unique cohesive and independently executable set of data movement types. Four data movement types are identified by ISO19761: an 'Entry' moves a data group into software from a functional user and an 'eXit' moves a data group out. Further, 'Write' and 'Read' move a data group to and from persistent storage, respectively. One (1) CFP (i.e. COSMIC Function Point) represent a functional size measurement of each counted data movement type [38].

The interactions within a software component describe its internal data movements. The first set of data movements is the set of interactions between the component and its attributes. For instance, a component can query (i.e. Read) or change (i.e. Write) one or more of its attributes. The second set of data movements represents interactions between a component and its subcomponents (i.e. Entry and eXit). A component can use some of its subcomponents to realize the functionality of software. Internal component interactions in COSMIC Function Points (CFP) equal to the arithmetic summation of data movements between its attributes and the data movements between its components.

Interactions of subcomponents are also considered, total number of interactions within a component are internal interactions added to the

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

interactions occurring inside all of the subcomponents. Because of a subcomponent is also a component; such definitions applied recursively. The number of component interactions in CFPs equal to the arithmetic summation of the number of internal component interactions and the number of sub-components interactions.

Software sub-components can be classified as related or unrelated sub-components. Related sub-components are those participating in internal component interactions. For instance, if a subcomponent use or depend-on attribute or another subcomponent, the subcomponent relates to its parent component or considered as unrelated. Once software sub-components are classified, it is possible to count component interactions and they added together. On the other hand, when counting interactions of related components, only subset of related subcomponents is considered. All of the sub-components interactions can also be counted, whether they are related or not. The total number of interactions for a set of components is equal to arithmetic summation of all components interactions calculated previously.

The measure of a cohesion on a ratio scale can take any value between zero and one. The cohesion ratio of a software component is the proportion of its related functionality (See Eq. 1). If a software component does not interact between its attributes and its sub-components, then cohesion ratio is zero. The cohesion ratio is undefined if there are no sub-components and no interactions between its attributes. When a component has no sub-components, cohesion ratio is set to a value of one, since the component forms a self-contained entity that is entirely independent. In addition, if all the interactions between components are related, then cohesion ratio raises up to a value of one. It is worth mentioning that software components should be located within the same software layer since different layers could have been developed using different types of technologies and therefore all functional size measurements should be calculated at the same level of granularity.

$$\text{Cohesion ratio} = \frac{\text{Internal components interactions} + \text{Related subcomponents interactions}}{\text{Internal components interactions} + \text{All subcomponents interactions}} \quad (1)$$

4. CASE STUDY: FIRST CLASS RELATION

4.1 Scope and Objective

This section presents an applicability verification of the proposed measurement model using structured specifications that adopt the concept of First Class Relation [39]. The proposed measurement model is used to measure the functional size of the first class relation exist between to two object-oriented class objects. Software engineers use class diagrams to represent the structure of object-oriented classes using attributes, methods and relationships; they use first class relations to represent the dependency (i.e. coupling between components) that might exist between such object-oriented classes. Figure 3 presents an example of between two object-oriented classes that interact with each other. The assigned objects are preserved references to instances of each other. If an object event added or deleted both references are updated accordingly. Class objects are required to specify the internal implementation details of the other object and which method of the other object to use in order to prevent an infinite loop.

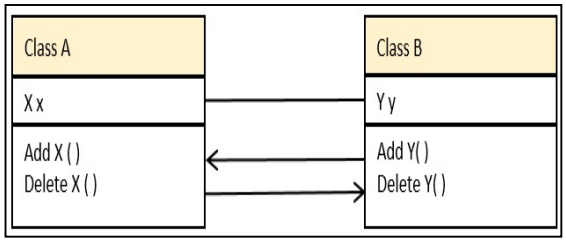

Fig 3 An example of two related object-oriented classes

4.2 Characterization of Measured Concepts

Figure 4 present coupling relationship between two object-oriented classes using the first class relation. Using first class relation, references to other class objects are not essential, and therefore this relationship certainly is no longer preserved inside class objects themselves.

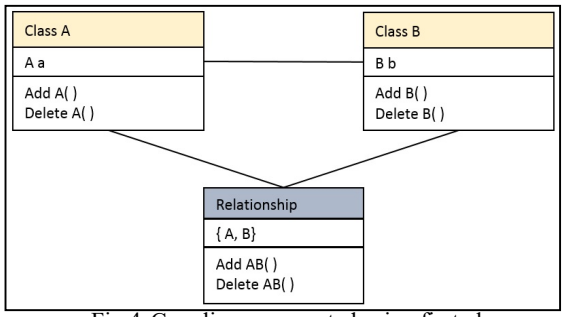

Fig 4 Coupling represented using first class relationship

For the purpose of this case study, the term "class object" is used to refer to the type of a specific object, and the term "instance object" is

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

used to refer to an instance of a specific class type. In addition, the term "relationship" is used to refer to an instance of a relation, in which a relationship typically consists of a set of tuples that include instance objects in a relation that are linked together or group of interacting instance objects [40].

4.3 Construction of Metamodel

Figure 5 presents the construction of the metamodel for first class relation divided into three sub-elements: state, preservation methods and cardinality. Construction is the bottle for object instances that participate in the relationship. The instance objects are part of the relationship at a given time and define the state of the relationship. Access to structure provided by preservation methods. Cardinality limits the number of instance objects on each side of the relationship.

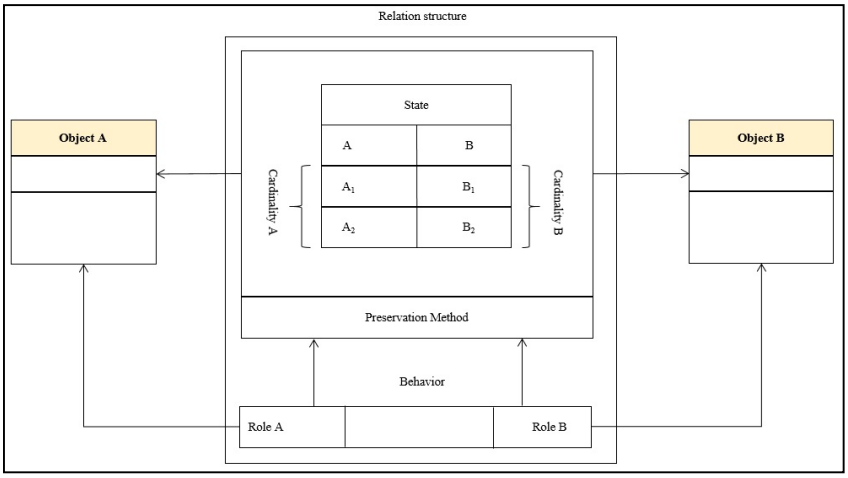

Fig 5 Construction of metamodel for first class relation

Another element to explain in the construction model is behavior (see figure 5). There are two types of behavior: active and reactive. Reactive behavior is activated by object instances that take part in the relation and producing additional instance objects in the relationships to react. This kind of behavior is typically implemented with the Spectator design pattern. Active behavior is initiated by third object (i.e. a client object). Roles are also part of behavior: roles describe the public interface of the objects that are used by relation once an object contributes in a relationship.

4.4 Numerical Assignment Rules

This part present the metamodel of first class relation mapped in accordance to the rules and concepts of the international standard for software functional size measurement ISO19761 [38].

This standardized method measures the functional size of a software product independently of the technology used to develop such a product, and based on the identified functional user requirements. The ISO19761 construct a generic model of software functional user requirements in order to clarify the boundary between hardware and software. In this model, software is typically bounded by hardware and it is used either by a human user or by an engineered device. The human user interacts with software using a variety of input/output devices. Furthermore, software is bounded by storage hardware such as RAM memory. The functionality of software is enclosed within the data groups of functional flows. In order to specify these functional flows, four data movement types are identified by ISO19761 as follows:

- Two data movement types (i.e. Entry and eXit) are identified to specify the functional flows between the human users and engineered devices from one side, and software from the other side.
- Two data movement types (i.e. Read and Write) are identified to specify the functional flows between storage and software.

Figure 6 presents the measurement metamodel of first class relation mapped in accordance to the rules and concepts of the international standard for software functional size measurement ISO19761. The specification of cohesion and coupling in software functional user requirements of a software development project is an instantiation of the proposed metamodel as presented in such figure.

Journal of Theoretical and Applied Information Technology
30th April 2018. Vol.96. No 8

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

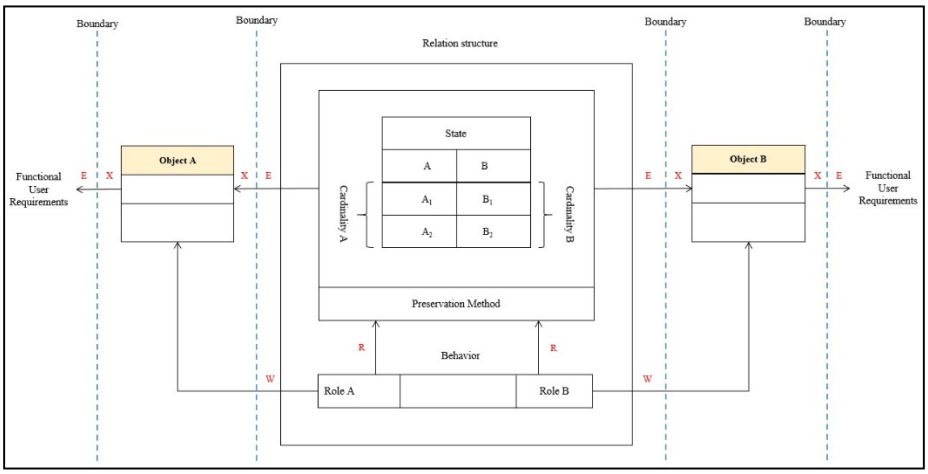

Fig 6 An Instance Measurement Metamodel Of First Class Relation

At an early phase of the software development life cycle, software engineers need to write the requirements specification document to specify the functional user requirements at a granularity level of movements of data groups. Then, cohesion and coupling requirements are directly measured using the proposed measurement model.

Table 2 presents functional size measurement of first class relation using the proposed measurement model. Four functional processes are identified (i.e. relational structure, roles, object A and B, and FUR), these functional processes are presented in column #1. On the other hand, column #2 presents data movement descriptions that moves data groups across the boundary. The corresponding type and number of data movements exist in each functional process is presented in column #3. For example, for a software relation structure, function 'a' relation structure sends a data group from 'a' state function to object A and B. Further, software role structure function 'Read' a data group from a preservation function and Write a data group to Object A and B. This corresponds to one Entry data movement type and one Exit data movement type, for a total functional size of two COSMIC Function Points (i.e. 2 CFPs). Therefore, the total functional size for the four identified functional processes yields twelve COSMIC Function Points.

Table 2: Functional Size Measurement Of The First Class Relation Using The Proposed Measurement Model

Functional Process	Data Movement Description	Data Movement Type
Relational Structure	Role A and B read from preservation method to give instructions to object A and B	2 Entry
	Role A and B write the instructional roles form preservation method to object A and B	2 eXit
Roles	Role A and B read from preservation method to give instructions to object A and B.	2 Read
	Role A and B write instructional roles form preservation method to object A and B	2 Write
Object A and B	Object A and B send a data movements to FUR	2 Entry
FUR	FUR receives data movements from object A and B	2 eXit
Total functional size measurement		**12 CFP**

4.5 Threats to Validity

An internal validity threat is associated with any changes in the design of this case study such as lack of description for the concepts to be evaluated in the case study. To mitigate the risk of this threat to validity, the principal researcher who proposed the measurement model has conducted experiment three weeks after completing the whole design of

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

the mentioned model.

An external validity threat is expressed as the extent that the experimental results can be generalized beyond the experimental settings. The proposed measurement model of cohesion and coupling is experimented using only the structural specifications of the first class relation. To mitigate the risk of this threat to validity, further studies should be conducted in the future using the requirements specifications of different software products of different types.

5. CONCLUSION

This paper proposed a new measurement model to measure the degree of cohesion and coupling exist within and between system components based on international ISO19761 international standard. The proposed measurement model measures the functional size interactions exist between components of software systems using the measurement concepts of ISO19761, and independently from development technology used to develop the software product. Four steps are conducted to build the measurement model; determination of measurement objective, followed by a characterization of cohesion and coupling terms. After that, cohesion and coupling metamodel is constructed, and finally an identification of numerical assignment rules is conducted for cohesion and coupling. The results of the case study shows that the proposed measurement model is capable of measuring the degree of cohesion and coupling exist between different components, in which coupling is using the First Class Relation in object-oriented structured specifications.

This measurement will help to improve the planning, management, and development of software at different phases of software life cycle. Further, the measurement results of the proposed model can be used for software benchmarking purposes conducted by specialized groups such as the International Software Benchmarking Group (ISBSG). Future work will be directed to conduct more case studies using requirements specifications of different software systems of different types, in order to generalize the results reported in this paper. In addition, future work will be directed to automate the measurement process to build an automated measurement tool.

REFERENCES

[1] J. Alghamdi, *"Measuring software coupling"*, Proceedings of the 6th international conference on software engineering, parallel and distributed systems, Corfu Island, Greece, 2007, pp. 6-12.

[2] M. Kiewkanya, P. Muenchaisri, *"Measuring maintainability in early phase using aesthetic metrics"*, Proceedings of 4th international conference on software engineering, parallel and distributed systems, Salzburg, Austria, 2005, pp. 1-6.

[3] D. Kushwaha, A. Misra, *"A complexity measure based on information contained in the software"*, Proceedings of 5th international conference on software engineering, parallel and distributed systems, Madrid, Spain, 2006, pp. 187-195.

[4] T. Lethbridge, N. Anquetil, *Experiments with coupling and cohesion metrics*. University of Ottawa, Ottawa, Canada. http://www.site.uottawa.ca/~tcl/papers/metrics/expwithCouplingCohesion.html (Accessed on February 3, 2018)

[5] S. Counsell, S. 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.

[6] P. Bourque, R.E. Fairley, "Guide to the software engineering body of knowledge (SWEBOK)", *IEEE Computer Society Press*, 2014, USA.

[7] A. Marcus, D. Poshyvanyk, "*The conceptual cohesion of classes*", Proceedings of the 21st IEEE international conference on software maintenance, Budapest, Hungary, 2005, pp. 133-142.

[8] E. B. Allen and T. M. Khoshgoftaar, "*Measuring coupling and cohesion: an information-theory approach*", Proceedings of the 6th International Software Metrics Symposium, Boca Raton, FL, USA, 1999, pp. 119-127.

[9] T. M. Meyers, D. Binkley, "An empirical study of slice-based cohesion and coupling metrics", *ACM Transactions on Software Engineering and Methodology*, Vol. 17, No. 1, 2017, pp. 1-27.

[10] W. Stevens, G. Myers and L. Constantine, "Structured design", *IBM Systems Journal*, Vol. 2, 1974, pp. 115-139.

[11] L. Badri, M. Badri and G. A. Badara,

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

"Revisiting class cohesion: an empirical investigation on several systems", *Journal of Object Technology*, Vol. 7, No. 6, 2008, pp. 55-75.

[12] L. C. Briand, S. Morasca and V. R. Basili, "Property-based software engineering measurement", *IEEE transactions on software engineering*, Vol. 22, No. 1, 1996, pp. 68-86.

[13] C. Zhenqiang, Y. Zhou, B. Xu, J. Zhao and H. Yang, "*A novel approach to measuring class cohesion based dependence analysis*", Proceedings of the international conference on software maintenance, Montréal, Canada, 2002, pp. 377-384.

[14] A. Marcus, D. Poshyvanyk, "*The conceptual cohesion of classes*", Proceedings of the 21st IEEE international conference on software maintenance, Budapest, Hungary, 2005, pp. 133-142.

[15] K. Byung-Kyoo, J. M. Bieman, 1996, "*Design-level cohesion measures: derivation, comparison, and applications*", Proceedings of 20th international computer software and applications conference, Seoul, Korea, 1996, pp. 92-97.

[16] S. Makela, V. Leppanen, 2007. *"Client based object-oriented cohesion metrics"*, 31st annual international computer software and applications conference, Beijing, China, 2007, pp. 743-748.

[17] I. W. Soares, L. Agner, P. Cézar Stadzisz, J. M. Simão, "Application of platform models in model driven engineering of embedded software", *Journal of Computer Science*, Vol. 11, No. 12, 2015, pp. 1075-1081.

[18] L. Agner, I. Soares, J. M. Simão, P. Cézar Stadzisz. 2014, "Applying black box testing to model transformations in the model driven architecture context", *Journal of Computer Science*, Vol. 10, No. 8, 2014, pp. 1423-1427.

[19] K.G. Madhwaraj, "Empirical comparison of two metrics suites for maintainability prediction in packages of object-oriented systems: a case study of open source software", *Journal of Computer Science*, Vo. 10, No. 8, 2014, pp. 1423-1427.

[20] B. Újházi, R. Ferenc, D. Poshyvanyk, T. Gyimóthy, *"New conceptual coupling and cohesion metrics for object-oriented systems"*, Proceedings of the 10th IEEE working conference on source code analysis and manipulation, Timisoara, Romania, 2010, pp. 33-42.

[21] N. Rajkumar, C. Viji, S. Duraisamy, "Measuring cohesion and coupling in object oriented system using java reflection", *ARPN Journal of Engineering and Applied Sciences*, Vol. 10, No. 7, 2015, pp. 3096-3101.

[22] A. Maheshwari, A. Tripathi, D. S. Kushwaha, *"A new design based software coupling metric"*, 14th International conference on information technology, Odisha, India, 2014, pp. 351-355.

[23] J. Ludwig, S. Xu, F Webber, "*Compiling static software metrics for reliability and maintainability from GitHub repositories*", IEEE International Conference on Systems, Man, and Cybernetics, Banff, Canada, 2017, pp. 5-9.

[24] S. Almugrin, A. Melton, "*Estimation of responsibility metrics to determine package maintainability and testability*", 2nd international conference on trustworthy systems and their applications, Hualien, Taiwan, 2015, pp. 100-109.

[25] S. Almugrin, W. Albattah, O. Alaql, M. Alzahrani, A. Melton, "*Instability and abstractness metrics based on responsibility*", IEEE 38th annual computer software and applications conference, Vasteras, Sweden, 2014, pp.364-373.

[26] C. Faragó, P. Hegedűs, G. Ladányi, R. Ferenc, "*Impact of version history metrics on maintainability*", 8th international conference on advanced software engineering and its applications, Jeju Island, Korea, 2015, pp. 30-35.

[27] A. Shafiabady, M. Mahrin, M. Samadi, "*Investigation of software maintainability prediction models*", 18th International Conference on advanced communication technology, Pyeongchang, South Korea, 2016, pp. 783-786.

[28] D. Gonzalez, J. Santos, A. Popovich, M. Mirakhorli, M. Nagappan, "*A large-scale study on the usage of testing patterns that address maintainability attributes: patterns for ease of modification, diagnoses, and comprehension*", Proceedings of the 14th International conference on mining software repositories, Buenos Aires, Argentina, 2017, pp. 391-401.

[29] A. Jain, S. Tarwani, A. Chug, "*An empirical investigation of evolutionary algorithm for software maintainability prediction*", IEEE students conference on electrical, electronics and computer science, Bhopal, India, 2016, pp.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

1-6.

[30] S. Rongviriyapanish, T. Wisuttikul, B. Charoendouysil, P. Pitakket, P. Anancharoenpakorn, and P. Meananeatra, "*Changeability prediction model for Java class based on multiple layer perceptron neural network*", 13th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology, Chiang Mai, Thailand, 2016, pp. 1-6.

[31] R. Mo, Y. Cai, R. Kazman, L. Xiao, Q. Feng, "*Decoupling Level: A new metric for architectural maintenance complexity*", IEEE/ACM 38th IEEE International Conference on Software Engineering, Austin, USA, 2016, pp. 499-510.

[32] B. S. Panca, S. Mardiyanto, B. Hendradjaya, "*Evaluation of software design pattern on mobile application based service development related to the value of maintainability and modularity*", International Conference on Data and Software Engineering, Denpasar, Indonesia, 2016, pp. 1-5.

[33] A. Baqais, M. Amro, M. Alshayeb, "*Analysis of the correlation between class stability and maintainability*", 7th International Conference on Computer Science and Information Technology, Amman, Jordan, 2016, pp. 1-4.

[34] M. Alshayeb, M. Naji, M. O. Elish, "Towards measuring object-oriented class stability", *IET Software Journal*, Vol. 5, No. 4, 2011, pp. 415-424.

[35] A. Abran, "Software metrics and software metrology", *IEEE Computer Society Press*. ISBN: 047059720897804705972000, 2010.

[36] Institute of Electrical & Electronics Engineers, "IEEE systems and software engineering vocabulary", *IEEE Computer Society Press*, New York, USA, 2010.

[37] International Organization for Standardization, "Information Technology - software measurement - functional size measurement Part 1: definition of concepts (ISO/IEC-14143-1)", *International Organization for Standardization*, Geneva, Switzerland, 2007.

[38] International organization for standardization, "ISO19761: a functional size measurement method: COSMIC", *International Organization for Standardization*, Geneva, Switzerland, 2013.

[39] G. Bierman, A. Wren, "First-class relationships in an object-oriented language – Technical Report UCAM-CL-TR-642", *Computer Laboratory, University of Cambridge*, ISSN: 1476-2986, United Kingdom, 2005.

[40] S. Balzer, T.R. Gross, P. Eugster, *"A relational model of object collaborations and its use in reasoning about relationships"*, The 21st European conference object oriented programming. Berlin, Germany, 2007, pp. 323-346.