A Comparison Tool For Quality Software Requirement Specification

Novi Trisman Hadi
Informatics Dept, Institut Teknologi Surabaya
Surabaya, Indonesia
Novi.trisman@itsu.ac.id

Ari Firmanto
Sekolah Tinggi Ilmu Komputer Bali Indonesia
Denpasar, Indonesia
firmanto@lecturer.pelitaindonesia.ac.id

***Abstract*— Most of the problems in software development come from a bad requirement specification. Failure on the requirements gathering phase is usually caused by unclear, ambiguous, inconsistent or incomplete requirements [1]. Thus, many researchers work on how to improve the quality of requirement specification. Even this is not the largest task of a project, it is really important to provide a flawless requirement specification**

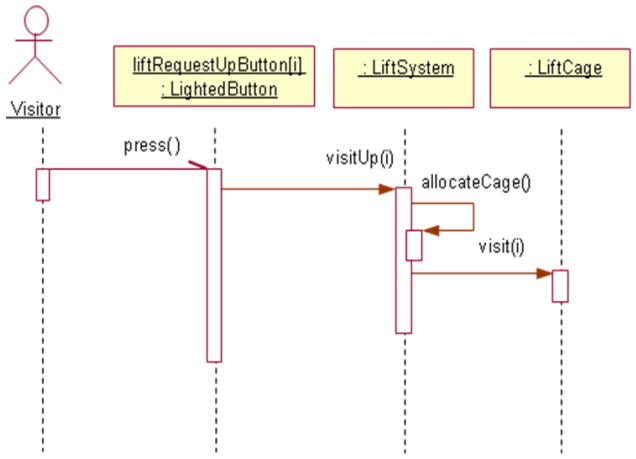
I. INTRODUCTION

Unified Modeling Language is a modeling language standard that has been known and used by software engineers for many years. It plays a main role within software development life cycle of a project [1,2]. The language allows designers to models the interaction between system and users, interaction between objects, behavior of objects, and implementation and logical structure of the system. These models represent different views and concerns of a single system. Throughout the life cycle, the models may change and evolve due to growing knowledge on the problem domain, lack of knowledge, skills, experience of designers, and constantly changing requirements. Differences between models may also be the result of change-propagation on models of the same software within versions, feature dissimilarities due to specific characteristics of different domains, and other aspects regarding project team attributes, such as experience and skills [3].

This study proposes a method to measure similarity between two different UML (Unified Modeling Language) sequence diagrams. The method was adopted from Al-Khiaty & Ahmed [4]. The result allows further reuse of software artifacts during the software development process. Thus, it enables software engineers to develop project not from scratch, but from an existing project of a similar design. The goal would be to improve efficiency within a software project.

This approach measured the structural and semantic similarity between two sequence diagrams. Principally, it recognizes semantic associations between attributes and structural similarity of the two diagrams [5, 6]. The similarity scores of the two diagrams indicate their propinquity. The semantic similarity draws on three sequence diagram attributes, i.e. class name, method invocation name, and message. The structural similarity draws on two sequence diagram attributes, i.e. neighborhood class name, fan-in, and fan-out. For both similarity measurements, this study employed natural language processing to pre-process each label of attributes.

There are two types of model similarity measurements [4]. The first one is exact similarity measurement. It aims at strictly measuring the exact similarity between two models or between two sub-sets or sub-components. It is not lenient to any alternatives that may exist between them. This type of similarity measurement is suitable for testing the compliance between artifacts or for detecting conflicts in sub-versioning system. The second one is inexact similarity measurement. It aims at loosely measuring the general similarity between two models, two sub-sets, or sub-components. It is lenient to any alternatives or minor variations that may exist between them. This type of similarity measurement is suitable for model retrieving in a recommendation system.



(M1)

Figure 1. A Sequence Diagrams that Implement Use Case 'Request Lift'

There are a number of previous researches that focused on developing methods to measure the similarity of two UML models. Dijkman et al. proposes three metrics for measuring similarity of process models, i.e. node, structural, and behavioral similarities [7]. A work by Al-Batran, Schatz, and Hummel improves normalization techniques in order to cover not only semantic structure but also behavior clones. They extends the model to its equivalent unique normal forms before being measured for similarity [8]. A work by Al-Khiaty & Ahmed introduces retrieving similar model from repository based on a query model. It focuses on UML class diagram [4] and uses a greedy algorithm for measuring the model similarity. Störrle introduces a model clone detection algorithm for UML models [9]. He suggests a formal definition of model clones and uses heuristic algorithm based indexing function to measure similarity between two models. His works only focus on four types of UML models, i.e. use case diagram, class diagram, activity diagram, and state diagram.

Our proposed solution is an adoption of the work introduced by Al-Khiaty and Ahmed [4]. They presented a framework for retrieving similar class diagram from a project repository by measuring their similarity. The similarity measurement comprises of various different elements, which includes lexical name, attributes, operations, internal, neighborhood similarity, and a combination of them. The similarity measurement utilizes a deterministic algorithm to discover the best matching element-pairs of the two class diagrams. Since their focus was the UML class diagram, the word 'model' henceforth refers to a UML class diagram, which represents the structural view of a software system. Additionally, the words "element" and "class" are used exchangeable in their approach.

Our work focuses on another behavioral view of UML diagrams, i.e. sequence diagrams. The paper is organized as follows. The second section introduces the approach we used to measure the semantic and structural similarity between two sequence diagrams. Then, the third section provides the test

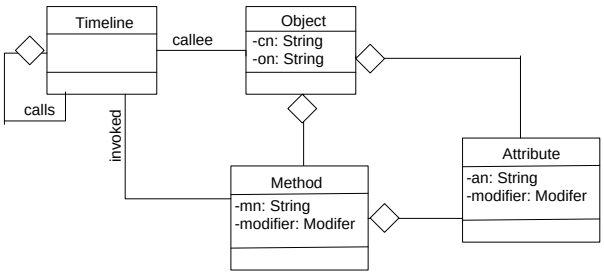


Figure 2. Metadata of Sequence Diagram

cases that we used in this research. The fourth section presents the results and analysis on the experimentation that we have conducted. The last section presents the conclusion and further work.

II. SIMILARITY MEASUREMENT METHOD

This section elaborates the adopted method that was modified for measuring two sequence diagrams. The method comprises of three different steps, i.e. diagram preprocessing, sequence diagram's attributes similarity measurement, and model similarity measurements.

A. Diagram Preprocessing

In order to measure the similarity of two sequence diagrams, each diagrams need to be preprocessed. Figure 1 illustrates a sequence diagram that implement a use case "Request Lift" on a Lift System. The diagram preprocessing extracts metadata of each sequence diagram into a sequence diagram metamodel. For this purpose, we introduce a metadata model for a sequence diagram as shown in Figure 2. The metamodel are formed as a set of components. The components can be grouped as two sets, i.e. objects and timelines. An object is an instance of a class. It has an object name and class name. The object has a set of methods and attributes. A method has a method name and modifier, i.e. private, public, or protected. An attribute has an attribute name and modifier.

A timeline is a sequence of method invocations, i.e. it is formed as a series of object-method pairs. The timeline contains an object callee and the method being invoked. The order of pairs reflects the sequence the pairs would be executed. A pair represents any event when an object calls a private method or a public method of another class. A timeline may have a set of other timelines being invoked as a consequence of the respected timeline invocation.

For extracting the metadata, an open source UML modeling tool was used. The tool converts each graphical model into an XMI-format diagram file. The XMI tags are parsed to produces the required sequence diagram metadata. Given M1, the extracted metadata may contain the following:

Objects

- *Object_0 {cn: Visitor}*
- *Object_1{on:LiftRequestButton, cn:RequestButton, mi_0{mn:press}}*
- *Object2 {cn:LiftSystem, mi_1{mn:visitUp, an_0{an:i}, mod: public}, mi_2 {mn:allocateCage, mod:private}}*
- *Object3 {cn:LiftCage, mi_3 {mn:visit, an_1{an:i},mod:public}}*

Timelines:

- *Object_0 → mi_0*

  Object_1 → mi_1

  Object_2 → mi_2

  Object_2 → mi_3

In M1, there are four objects, *Object_0, Object_1, Object_2*, and *Object_3*. An object in sequence diagram may contain a class name, an object name, and one or more methods). The label *on* represents the name of an object. For example, Object_1 is labeled with object name "LiftRequestButton". The label *cn* represents the class name of an object. For example, Object_1 is an instantiation of LiftRequest class; therefore, its *cn* is LiftRequeest. Object_1 has one method invocation, i.e. mi_1. A method may contain a method name, a returned data type, one or more arguments, and access modifier. The label *mn* represents the name of a method. For example, *mi_1* is labeled with method name "press". An argument may contain an argument name and argument data type. The label *an* represents the name of an argument. For example, an_1 is labeled with argument name "i". The label *ad* represents the data type of an argument.

Aside from the object, *M1* contains four timelines, where timeline 1 is an event where *Object_1* invokes *mi_0*. It also has a child timeline, i.e. *Object_1* invokes *mi_1*. It means that in the body of method LiftRequestButton.press(), there is a line that calls LiftSystem.visitUp(). Thus, the second timeline has two children, i.e. *Object_2* invokes *mi_2* and *Object_2* invokes *mi_3*. Given the above description, we can also extract the following metadata from M2 as shown in Figure 3. The extracted metadata can be described as follow.

Objects

- *Object_0 {cn: Visitor}*
- *Object_1{on:LiftRequestButton, cn:RequestButton, mi_0{mn:press}}*
- *Object2 {cn:LiftSystem, mi_1{mn:visitUp, an_0{an:i}, mod: public}, mi_2 {mn:allocateCage, mod:private}}*
- *Object3 {cn:LiftCage, mi_3 {mn:visit, an_1{an:i},mod:public}}*

Timelines:

- *Object_0 → mi_0*

  Object_1 → mi_1

  Object_2 → mi_2

  Object_2 → mi_3

B. Similarity Measurements

Not like Khiaty and Ahmed [4], the similarity measurement of sequence diagram utilizes two types of similarity information, i.e. structural and message-sequence information. Nevertheless, in the proposed method, the semantic similarity resided in both types of information is also explored. The message sequence information is used to measure the sequences (behaviors) of the two sequence diagrams. The semantic information is used to measure the labeling similarity between components of the two sequence diagrams. The structural information is used to measure the similarity between components of objects that needed to realize the intended use case. The object components include object and class name, attributes, and methods. Each types of similarity will be aggregated to measure the degree of similarity between the two sequence diagrams. Since each type may have different impact on the total similarity score, we introduced weights. The semantic similarity between two sequence diagrams is calculated by using a word similarity thesaurus on labels of objects, attributes, and methods or messages.

As already mentioned, there are two set of similarity metrics, i.e. structural similarity (*strucSim*) and message sequence similarity (*msSim*). The strucSim measures the semantic similarity between classes resides in the two sequence diagrams (i.e. d_1 and d_2) as specified in equation (1).

$$strucSim(d_1, d_2) = Max\left(\sum_{i,j=1}^{Max(|O_1|,|O_2|)} oSim(o_i, o_j)\right) / (|O_1| + |O_2|) \quad (1)$$

where O_1 and O_2 is the list of object resides in sequence diagram d_1 and d_2, respectively. Object similarity of two objects, $oSim(o_1,o_2)$, is the semantic similarity between two objects as specified in (2), where $o_1 \epsilon O_1$ and $o_2 \epsilon O_2$.

$$oSim(o_1, o_2) = w_c \times cSim(o_1, o_2) + w_m \times mSim(o_1, o_2) \quad (2)$$

where w_c *and* w_m are arbitrary weights assigned to class similarity (cSim) and method similarity (mSim), respectively. The class similarity measures the semantic similarity of object names and or class name between the two objects, o_1, o_2. It collects similarity of tokens from the tokenized strings of the two objects and calculate cosine similarity values for the two objects [10].

$$cSim(o_1, o_2) = Max\left(\sum_{i,j=1}^{Max(|CP_1|,|CP_2|)} CosineSim(cp_i, cp_j)\right) / (|CP_1| + |CP_2|) \quad (3)$$

where CP_1 *and* CP_2 are tokenized strings of class and object names of o_1 and o_2, respectively. The method similarity measures the semantic similarity of methods' properties between the two objects, o_1, o_2., by calculating their cosine similarity. The properties include method's name and arguments.

$$mSim(o_1, o_2) = Max\left(\sum_{i,j=1}^{Max(|M_1|,|M_2|)} dmSim(m_i, m_j)\right) / (|M_1| + |M_2|) \quad (4)$$

where M_1 *and* M_2 are a set of methods of o_1 and o_2, respectively. The detail method similarity (dmSim) measures the semantic similarity between two methods, m_1, m_2. The detail method similarity of two methods, $dmSim(m_1,m_2)$, is the semantic similarity between two methods as specified in (6), where $m_1 \epsilon M_1$ and $m_2 \epsilon M_2$.

$$dmSim(m_1, m_2) = w_{mn} \times WuP(mn_1, mn_2) + w_{ma} \times aSim(m_1, m_2) \quad (5)$$

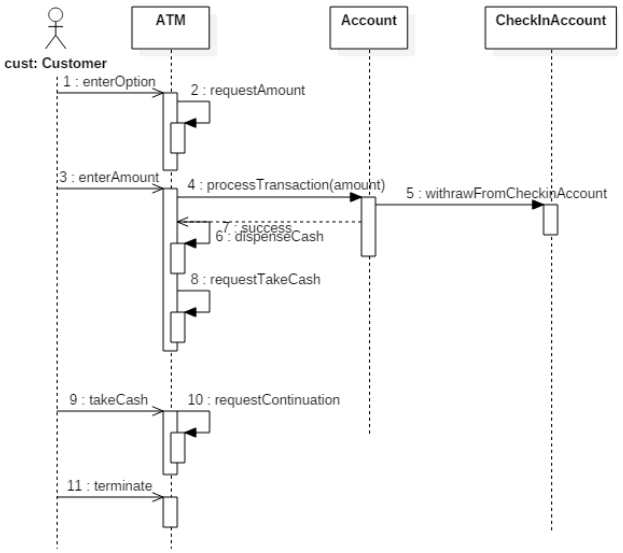


Figure 3 Sequence Diagram of 'Withdraw Money' (SD1) of Project-1.

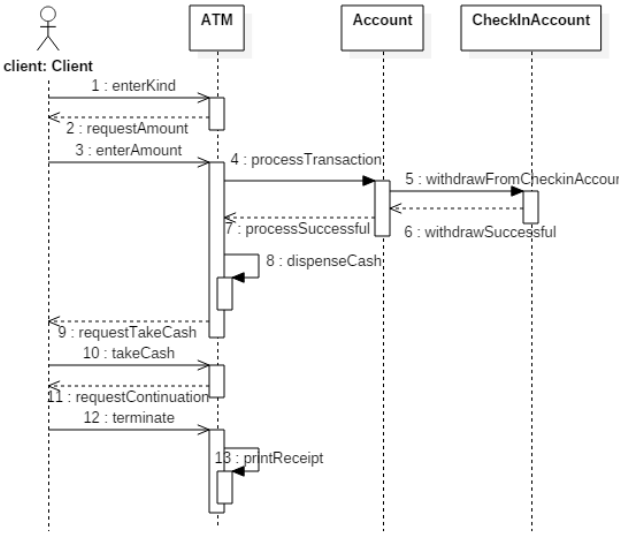


Figure 4 Sequence Diagram of 'Withdraw Money' (SD2) of Project-2.

where w_{mn} *and* w_{ma} are arbitrary weights assigned to name similarity of method m_a and m_2, and attribute similarity (aSim) of method's attribute properties, respectively. The attribute similarity measures the semantic similarity of attributes' properties between the two objects, o_1, o_2, by calculating their cosine similarity. The properties include name and type.

$$aSim(o_1, o_2) = Max\left(\sum_{i,j=1}^{Max(|A_1|,|A_2|)} CosineSim(a_i, a_j)\right) / (|A_1| + |A_2|) \quad (6)$$

where A_1 *and* A_2 are a set of attributes of o_1 and o_2, respectively. The similarity value of two method names is

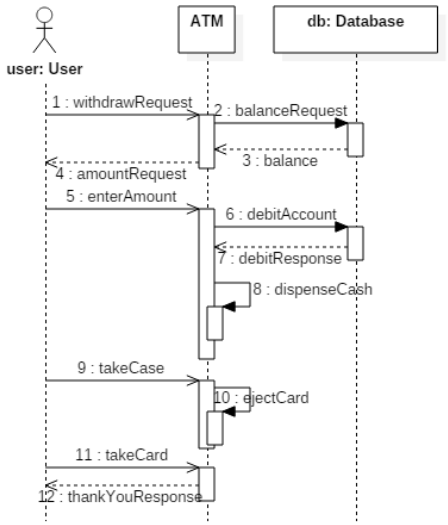


Figure 5 Sequence Diagram of 'Withdraw Money' (SD3) of Project-3.

calculated as mean value of the semantic path length of the two names (*WuP*) [10]. If the mean value is equal to zero, than it is calculated as a difference between two sequences of string.

While *strucSim* measures structural similarity between classes in the two diagrams, The *msSim* measures the similarity between two sequences of messages passed between classes within the two diagrams. Equation 7 shows how to calculate the msgSim.

$$msSim(d_1, d_2) = Max\left(\sum_{i,j=1}^{Max(|MS_1|,|MS_2|)} msgSim(ms_i, ms_j)\right) / (|MS_1| + |MS_2|) \quad (7)$$

where MS_1 and MS_2 is the sequence of messages invoked during the realization of use case as stated in sequence diagram d_1 and d_2, respectively. Similarity of two messages, $msgSim(ms_1, ms_2)$, is the semantic similarity between two messages as specified in (8), where $ms_1 \epsilon MS_1$ and $ms_2 \epsilon MS_2$.

$$msgSim(ms_1, ms_2) = w_{ms1} \times cSim(src_1, src_2) + w_{ms2} \times dmSim(m_1, m_2) + w_{ms3} \times cSim(dst_1, dst_2) \quad (8)$$

where $w_{ms1}, w_{ms2}, and\ w_{ms3}$ are arbitrary weight assigned to source class name (src_i), method name (m_i), and destination class name (dst_2) similarities, respectively.

Given the similarity values of object properties and message sequences of the two diagrams, we can measure the similarity of the two diagrams by agregating both similarity values. Equation 9 shows the similarity values of two diagrams.

$$seqSim(d_1, d_2) = w_{os} \times strucSim(d_1, d_2) + w_{ms} \times msSim(d_1, d_2) \quad (9)$$

where $w_{os}, and\ w_{ms}$ are arbitrary weight assigned to structural similarity and message sequence similarities, respectively.

III. EMPERICAL RESULT AND ANALYSIS

The main objective of this study is to compare and then select the best set of metrics, i.e. lexical information, internal information, and neighborhood information, to measure the similarity between two sequence diagrams. In order to provide initial indication on the possibility to use the set of similarity measurement metrics, we have conducted experimentation.

Table 1. Diagram Similarity Between the Three Sequence Diagrams

Diagram-pairs	StrucSim	SeqSim	Avg
SD1-SD2	0.851	0.81	0.84
SD1-SD3	0.494	0.65	0.53
SD2-SD3	0.561	0.65	0.58

Table 2. Object Similarity between SD1 and SD2

oSim	o2_1	o2_2	o2_3	o2_4
o1_1	0.75	0.06	0.04	0.07
o1_2	0.11	0.92	0.30	0.21
o1_3	0.04	0.26	0.73	0.61
o1_4	0.09	0.21	0.51	1.00

Table 3. Object Similarity between SD1 and SD3

oSim	o3_1	o3_2	o3_3
o1_1	0.70	0.06	0.03
o1_2	0.07	0.82	0.28
o1_3	0.04	0.25	0.45
o1_4	0.08	0.19	0.43

Table 4. Object Similarity between SD2 and SD3

oSim	o3_1	o3_2	o3_3
o2_1	0.95	0.06	0.03
o2_2	0.07	0.74	0.27
o2_3	0.04	0.38	0.55
o2_4	0.08	0.19	0.44

The experimentation compares two sequence diagrams of different projects. We measured the similarity of sequence diagram pairs from the same application domain, i.e. automatic teller machine. There are three sequence diagrams. All of them model the sequence of object interactions that realize a 'withdraw money' use case of Automatic Teller Machine (ATM) system. Figure 3-5 shows the description of three different sequence diagrams that implement 'withdraw money' use case.

We measured the similarity of each pair of sequence diagrams, i.e. SD1-SD2, SD1-SD3, and SD2-SD3. Using equation 7 and by setting the result of measuring the similarities between the three diagrams can be calculated. Table 1 shows that SD1- SD2 pair has the highest similarity values. Both SD1 and SD2 have relatively the same similarity values to SD3. The weight of $w_{os}\ and\ w_{ms}$ are set experimentally to 0.8 and 0.2, respectively

The similarity values of SD1-SD2 are calculated based on the object property and message sequence similarities of SD1 and SD2. Table 2 shows the object similarity between lifeline in SD1 and SD2. SD1 has four lifelines, i.e. Customer (o1_1), ATM (o1_2), Account (o1_3), and CheckInAccount (o1_4). SD2 also four lifelines, Client (o2_1), ATM (o2_2), Account (o2_3), and CheckInAccount (o2_4). The weight of $w_c\ and\ w_m$ are set experimentally to 0.2 and 0.8, respectively. Although the class names of objects in both diagrams are lexically the same, we can see that not all best object pairs are considered 100% similar. This is because the structures of the lexically similar objects are actually different. By using equation 1, the structure similarity of SD1-SD2 pair is 0.851. The yellow cells are the best set of object-pair values given SD1 and SD2 diagrams. We can see that object classes CheckInAccount, which are both in SD1 and SD2, are the most similar objects. Additionally, we can observe that the proposed equation is able to get the best-matched object-pairs. The proposed method also indicates that the more unbalance the number of objects resided in each diagram, the less similar

Table 5. Message Sequence Similarity between SD1 and SD2

	Client-enterKind-ATM	ATM-requestAmount-ATM	Client-enterAmount-ATM	ATM-processTransaction-Account	Account-withdrawFromCheckInAccount-CheckInAccount	CheckInAccount-withdrawSuccessfull-Account	Account-processSuccessful-ATM	ATM-dispenseCash-ATM	ATM-requestTakeCase-Client	Client-takeCash-ATM	ATM-requesContinuation-Client	Client-terminate-ATM	ATM-printReceipt-ATM
Customer-enterOption-ATM	0.854	0.436	0.789	0.349	0.303								
ATM-requestAmount-ATM	0.481	0.960	0.414	0.682	0.409								
Customer-enterAmount-ATM	0.834	0.515	0.909	0.268	0.285								
ATM-processTransaction(amount)-Account	0.295	0.512	0.360	0.923	0.508	0.373							
Account-withdrawFromCheckInAccount-CheckInAccount				0.530	0.920	0.511	0.472						
Account-success-ATM					0.418	0.517	0.737	0.316					
ATM-dispenseCash-ATM						0.257	0.416	0.960	0.744				
ATM-requestTakeCase-ATM							0.480	0.654	0.799	0.619	0.639		
Customer-takeCash-ATM									0.442	0.909	0.317	0.632	0.470
ATM-requesContinuation-ATM						0.269				0.476	0.799	0.442	0.669
Customer-terminate-ATM										0.632	0.125	0.909	0.350

Table 6. Message Sequence Similarity between SD2 and SD3

	User-withdrawRequest-ATM	ATM-balanceRequest-Database	Database-balance-ATM	ATM-amountRequest-User	User-enterAmount-ATM	ATM-debitAccount-Database	Database-debitResponse-ATM	ATM-dispenseCash-ATM	User-takeCash-ATM	ATM-ejectCard-ATM	User-takeCard-ATM	ATM-thankyouResponse-User
Client-enterKind-ATM	0.67	0.35	0.38	0.19	0.88							
ATM-requestAmount-ATM	0.54	0.70	0.46	0.74	0.48							
Client-enterAmount-ATM	0.60	0.31	0.40	0.35	0.93							
ATM-processTransaction-Account	0.34	0.59	0.33	0.51	0.35	0.58						
Account-withdrawFromCheckInAccount-CheckInAccount				0.34	0.28	0.44	0.43					
CheckInAccount-withdrawSuccessfull-Account			0.31		0.23	0.36	0.44	0.22				
Account-processSuccessful-ATM						0.39	0.55	0.39	0.43			
ATM-dispenseCash-ATM							0.41	0.87	0.52	0.58	0.39	
ATM-requestTakeCase-Client								0.52	0.46	0.52	0.40	0.48
Client-takeCash-ATM								0.44	0.93	0.44	0.81	0.27
ATM-requesContinuation-Client								0.44	0.34	0.44	0.39	0.66
Client-terminate-ATM										0.33	0.66	0.23
ATM-printReceipt-ATM										0.66	0.51	0.56

they become

Beside from calculating the structural similarity between SD1 and SD2, we have to calculate the message sequences in SD1 and SD2. Table 5 and 6 shows the result of calculating the message similarity between messages in SD1 and SD using equation 8. The weight of w_{ms1}, w_{ms2}, and w_{ms3} are set experimentally to 0.3, 0.5, and 0.2, respectively. Then using equation 9, we can calculate the message sequence similarity between SD1 and SD2 diagram, i.e. 0.74. Given this result, we can observed that the proposed equation able to get the best matched message-pairs. The proposed method also suggests that the more unbalance the number of messages resided in each diagram, the lesser similar they become.

The proposed sequence diagram similarity measurements shows that for most similar message sequences tends to shape a straight diagonal line. The lesser similar diagram pairs tend to shape curving lines. This pattern may be used to visually identify the dissimilarity between two sequence diagrams.

IV. CONCLUSION

This paper introduces an method for measuring similarity between UML sequence diagrams. The algorithm basically adopts the greedy approach in finding the best set of element-pairs. The proposed method considers two elements of the UML sequence diagram, i.e. object structure and message sequence. The paper also shows an early experimentation of the proposed method on a set of sequence diagrams of the same problem. The initial investigation indicates that the the structural similarity and message sequence similarity could be a good parameter in assessing the UML sequence diagram similarity. The best set of message sequence pairs of two sequence diagrams may be used as an indicator to visually identify the existance of insimilarity between the two sequence diagrams.

Further study should be carried out in order to answer several research questions. First, what would be the best weight settings the ensure the accuracy of similarity measurements. Second, how well the proposed method measure the similarity between UML sequence diagrams that realizes different use case from across domains. Finally, how well the performance of greedy approach used in this study compared with other approaches, such as simulated annealing and genetic algorithms. The performance includes computation complexity and finding the best set of pairs.

Acknowledgment

The research is in cooperation between Institut Teknologi Sepuluh Nopember and STIKOM Pelita Indonesia.

References

[1] D. Siahaan and F. Irhamni, "Advanced methodology for requirements engineering technique solution (AMRETS)," *Int. J. Adv. Comput. Technol.*, vol. 4, no. 5, pp. 75–80, 2012.

[2] D. Siahaan, *Analisa Kebutuhan dalam Rekayasa Perangkat Lunak*, 1st ed. Yogyakarta: Penerbit Andi, 2012.

[3] M. Chechik, S. Nejati, and M. Sabetzadeh, "A relationship-based approach to model integration," *Innov. Syst. Softw. Eng.*, 2011.

[4] M. A.-R. M. Al-Khiaty and M. Ahmed, "Similarity assessment of UML class diagrams using a greedy algorithm," in *Computer Science and Engineering …*, 2014.

[5] D. S. Kolovos, D. Di Ruscio, A. Pierantonio, and R. F. Paige,

"Different models for model matching: An analysis of approaches to support model differencing," in *2009 ICSE Workshop on Comparison and Versioning of Software Models*, 2009.

[6] K. Müller and B. Rumpe, "A Model-Based Approach to Impact Analysis Using Model Differencing," *Proc. 8th Int. Work. Softw. Qual. Maintainab.*, 2014.

[7] R. Dijkman, M. Dumas, B. Van Dongen, K. Reina, and J. Mendling, "Similarity of business process models : Metrics and evaluation," *Inf. Syst.*, 2011.

[8] B. Al-Batran, B. Schaetz, and B. Hummel, "Semantic Clone Detection for Model-Based Development of Embedded Systems," in *MODEL DRIVEN ENGINEERING LANGUAGES AND SYSTEMS*, 2011.

[9] H. Störrle, "Towards clone detection in UML domain models," *Softw. Syst. Model.*, 2013.

[10] D. Siahaan and S. Christina, *Using semantic similarity for identifying relevant page numbers for indexed term of textual book*, vol. 516. 2015.