Journal of Software Engineering and Applications, 2015, 8, 201-210
Published Online April 2015 in SciRes. http://www.scirp.org/journal/jsea
http://dx.doi.org/10.4236/jsea.2015.84021

ADTEM-Architecture Design Testability Evaluation Model to Assess Software Architecture Based on Testability Metrics

Amjad Hudaib, Fawaz Al-Zaghoul, Maha Saadeh, Huda Saadeh

Department of Computer Information Systems, King Abdullah II for Information Technology Department, The University of Jordan, Amman, Jordan
Email: Ahudaib@ju.edu.jo, Fawaz@ju.edu.jo, Saade_m87@ju.edu.jo, Hsaadeh@uop.edu.jo

Received 26 March 2015; accepted 20 April 2015; published 21 April 2015

Abstract

Architectural design is a crucial issue in software engineering. It makes testing more effective as it contribute to carry out the testing in an early stage of the software development. To improve software testability, the software architect should consider different testability metrics while building the software architecture. The main objective of this research is to conduct an early assessment of the software architecture for the purpose of its improvement in order to make the testing process more effective. In this paper, an evaluation model to assess software architecture (Architecture Design Testability Evaluation Model (ADTEM)) is presented. ADTEM is based on two different testability metrics: cohesion and coupling. ADTEM consists of two phases: software architecture evaluation phase, and component evaluation phase. In each phase, a fuzzy inference system is used to perform the evaluation process based on cohesion and coupling testing metrics. The model is validated by using a case study: Elders Monitoring System. The experimental results show that ADTEM is efficient and gave a considerable improvement to the software testability process.

Keywords

Software Testability, Testability Metrics, Software Architecture Evaluation, Software Cohesion, Software Coupling, Fuzzy Inference System

1. Introduction

Researches show that more than forty percent of software development efforts are spent on testing, with the

How to cite this paper: Hudaib, A., Al-Zaghoul, F., Saadeh, M. and Saadeh, H. (2015) ADTEM-Architecture Design Testability Evaluation Model to Assess Software Architecture Based on Testability Metrics. *Journal of Software Engineering and Applications*, **8**, 201-210. http://dx.doi.org/10.4236/jsea.2015.84021

percentage for testing critical systems being even higher [1] [2]. Many studies have been conducted to address software testing to reduce software testing efforts by handling software testability at early stages in the software development life cycle (SDLC) [1] [3]-[6].

Software Testability (ST) is the quality attribute about the easiness degree to which system's defects can be detected at testing phase [3]. The better the testability of particular software, the lower the testing effort required in testing phase. Moreover, the earlier the testability is considered, the lower the cost of testing phase. Thus, ST should be considered at the early phases of software development life cycle (SDLC), specifically in the software architecture (SA) [2] [3].

Software Architecture (SA) provides a high level of abstraction of software represented by software components and the connections between them. Software architect should consider the testability metrics while building the SA in order to improve software testability.

In this paper, we present a model (Design Testability Evaluation Model (ADTEM)) to conduct an early evaluation of the software architecture in order to make the testing process more effective.

ADTEM assesses the SA based on cohesion and coupling testability metrics. Fuzzy inference system is used in the evaluation process to assess the testability of SA according to a proposed set of inference rules. The model is implemented and applied on a case study (Elders Monitoring System) to ensure its applicability and effectiveness. The main activities of this model are:

1. Assess software system testability based on cohesion and coupling testability metrics.
2. Assess software system components testability based on cohesion and coupling testability metrics.
3. Classify software system components based on their level of testing effort.
4. Highlight the set of components in SA that should be improved to increase ST.

The rest of this paper is organized as the following: Section 2 summarizes some related works and discusses how they differ from the proposed model. The proposed model is discussed in details in Sections 3, 4 and 5. In Section 6, the proposed model is applied on a case study. Finally, Section 7 concludes the paper and discusses possible future work.

2. Related Work

Some studies focus on assessing system testability based on object-oriented metrics derived from system design [6]-[8]. In [4] a set of object-oriented metrics are defined to assess the testability of classes in a Java system. The relationships between these metrics and the testing process are evaluated. These metrics are: depth of inheritance tree, fan out, number of children, number of fields, number of methods, response for class, and weighted methods per class. Another class based testability metrics is proposed in [5]; which detects potential testability weaknesses of a UML class diagram and points out parts of the design that need to be improved to reduce the testing effort. In addition, it proposes a methodology for improving design testability. In [6], statistical analysis techniques, mainly correlation and logistic regression, are used in order to evaluate the capability of lack of cohesion metric to predict testability of object oriented classes collected from two java software systems.

Chidamber and Kemerer (CK) proposed a set of object oriented testability metrics [8]. The CK metrics were evaluated in [7] [9] [10], to investigate the relationship between these metrics and unit testing. A CK metrics are also used as a base for a fuzzy inference system to assess the testability efforts [1] [11]-[13].

In [14], object-oriented modeling techniques are used to help the product development team on the generation of the best fitted hardware/software architecture for a given problem under given constraints. It uses object and system testability metrics. The main focus of this work is to select near-optimum clustering of methods and attributes into objects with moderate cohesion and coupling. However, this approach applied to Data Flow Diagrams (DFD) that does not express information about process hierarchy.

In this paper, the evaluation is based on coupling and cohesion testability metrics that are derived from the software architecture. Both software system and software system components are considered in the evaluation process.

3. Proposed Model

In this paper, SA evaluation model is proposed. The evaluation is based on cohesion and coupling testability metrics. The fuzzy inference system is used to perform the evaluation process based on these metrics. The proposed approach consists of two phases (as shown in **Figure 1**). In the first phase, software system architec-

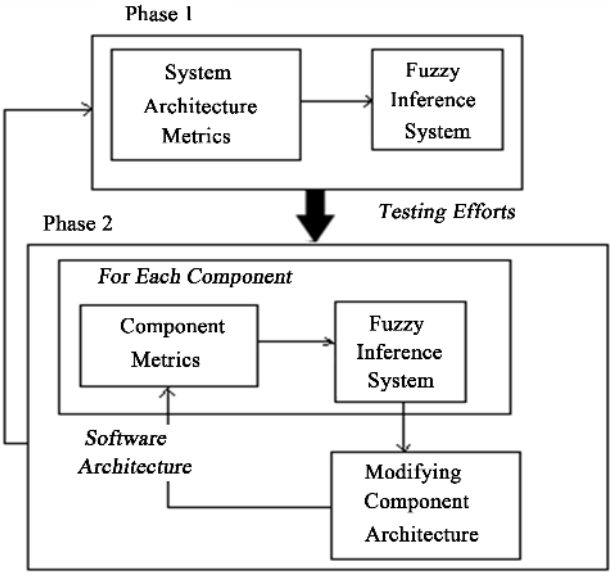


Figure 1. Proposed model.

ture is assessed according to the studied testability metrics (is discussed in Section 4). This assessment gives an indication on the required testing efforts for the entire software system. If software system required significant testing efforts, the second phase is performed to help in improving software system testability and reducing testing efforts. Moreover, in this phase, each component in the SA is assessed using the fuzzy inference system. Components' assessment specifies the testing efforts required for each component. After assessing all components, the architect can tell which component(s) required more effort for testing. Then, this component is modified to increase its testability; consequently, the whole software system testability. This process can be iterated as long as the system and components functionalities are maintained. In the following sections, the proposed model is discussed in details.

4. Software Architecture Testability Metrics

Two testability metrics: cohesion and coupling, are applied on architectural design. In the next subsections, these metrics are computed for each component as well as for the entire system.

4.1. Cohesion

The component cohesion (CCoh) is calculated for the system architecture to compute testing efforts. CCoh is the connectivity between system component, and it represents the component connectivity [3]. It is calculated by Equation (1); where the ratio between the numbers of component edges (E) and the maximum connectivity of the system (MC) is calculated [14]. The maximum connectivity corresponds to a complete graph which the total number of component is calculated by Equation (2).

$$\mathrm{CCoh} = \left(\frac{E}{\mathrm{MC}} \right) \tag{1}$$

$$\mathrm{MC} = \frac{1}{2} \times n(n-1) \tag{2}$$

where, n is the number of component. If $n = 1$, then CCoh = 1.

System cohesion (SCoh) is the average of the summation of components' cohesion, and it is calculated using Equation (3).

$$\mathrm{SCoh} = \left(\frac{\sum_{i=1}^{n} \mathrm{CCoh}(i)}{n} \right) \tag{3}$$

where, CCoh(i) is the cohesion of the i^{th} component.

4.2. Coupling

Coupling describes the interconnection between system components. Both component coupling (CCop) and system coupling (SCop) are considered. Component coupling can be described as the relation between component i and other components in the system as the number of fan-in and fan-out. It is calculated by Equation (4); as the ratio between the number of edges interconnecting component i and the total number of edges within system architecture [14].

$$\mathrm{CCop} = \frac{\sum_{i=1}^{n-1} e(i)}{E} \tag{4}$$

where n is the number of components, $e(i)$ is the number of edges of component i, and E is the total number of edges in the system architecture.

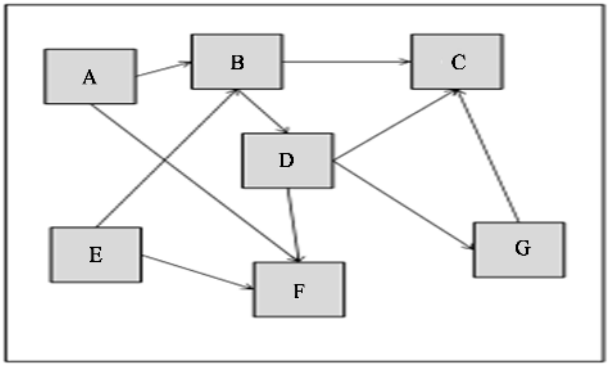
In addition, component coupling describes the component dependence (CDep), which is the number of components that depend on component i. CDep is calculated using Equation (5).

$$\mathrm{CDep} = \frac{\text{The ith component dependency tree}}{n} \tag{5}$$

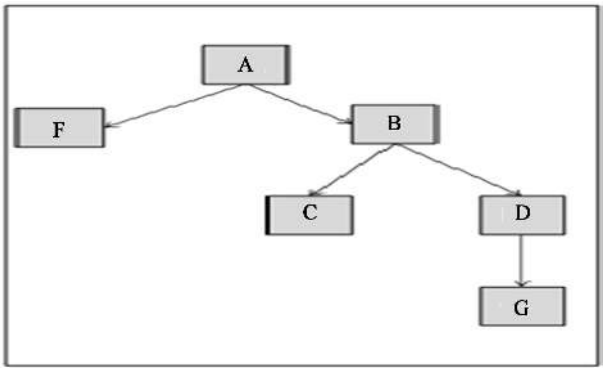
where the ith component of dependency tree is the tree rooted at component i. All other components that depend on component i are its children and its descendants.

Figure 2 shows the dependency tree in which **Figure 2(a)** presents the original architecture, and **Figure 2(b)** presents the corresponding dependency tree for component A.

System coupling can be described as the Cumulative Component Dependency (CCD) [15] which calculates the dependences for all components in the system using Equation (6).



(a)



(b)

Figure 2. An example on dependency tree. (a) The original architecture; (b) The corresponding dependency tree for component A.

$$\text{CCD} = \frac{\sum_{i=1}^{n} \text{The ith component dependency tree}}{n^2} \tag{6}$$

The interconnection between system components (SCop) is calculated by Equation (7).

$$\text{SCop} = \frac{\sum_{i=1}^{n-1} \text{CCop}}{n} \tag{7}$$

where n is the number of components in the system.

5. Fuzzy Inference System

The Mamdani fuzzy inference system is used to control the model and combine it with a set of linguistic control rules. The Mamdani uses three inputs and one output inference system. Mamdani inference system consists of three steps represented in **Figure 3**. In input fuzzification step, the value of each input variable is mapped to a fuzzy value according to the corresponding membership function. The second step is rules evaluation and aggregation. In this step, input fuzzy values are evaluated according to the inference rules which are listed in **Table 1**, where values: L is Low; M is Moderate; H is High; VL is Very Low; and VH is Very High. Then, all matched rules are aggregated to generate the fuzzy output value. The third step is the defuzzification step where the output is mapped from fuzzy domain to an output value.

The first input variable is the cohesion (Coh). It could be CCoh or SCoh. It has three values; low, medium, and high, each has a membership function illustrated in **Figure 4(a)**. The second input variable is the coupling (Cop), which could be CCop or SCop. Cop has three values; low, medium, and high. The corresponding membership functions are illustrated in **Figure 4(b)**. Finally, the third input variable is the dependency (Dep), which could be CDep or SDep. Dep is also has low, medium, and high values. The corresponding membership function

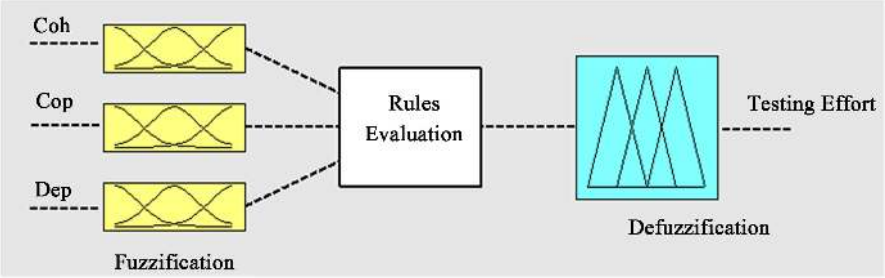


Figure 3. Fuzzy inference system steps.

Table 1. Fuzzy inference rules. Output values: L is Low; M is Moderate; H is High; VL is Very Low; and VH is Very High.

Coh	Cop	Dep		
		Low	Moderate	High
Low	**Low**	M	M	H
Medium	**Low**	L	M	M
High	**Low**	VL	VL	L
Low	**Medium**	M	H	VH
Medium	**Medium**	M	H	H
High	**Medium**	VL	M	H
Low	**High**	H	VH	VH
Medium	**High**	M	H	H
High	**High**	L	H	H

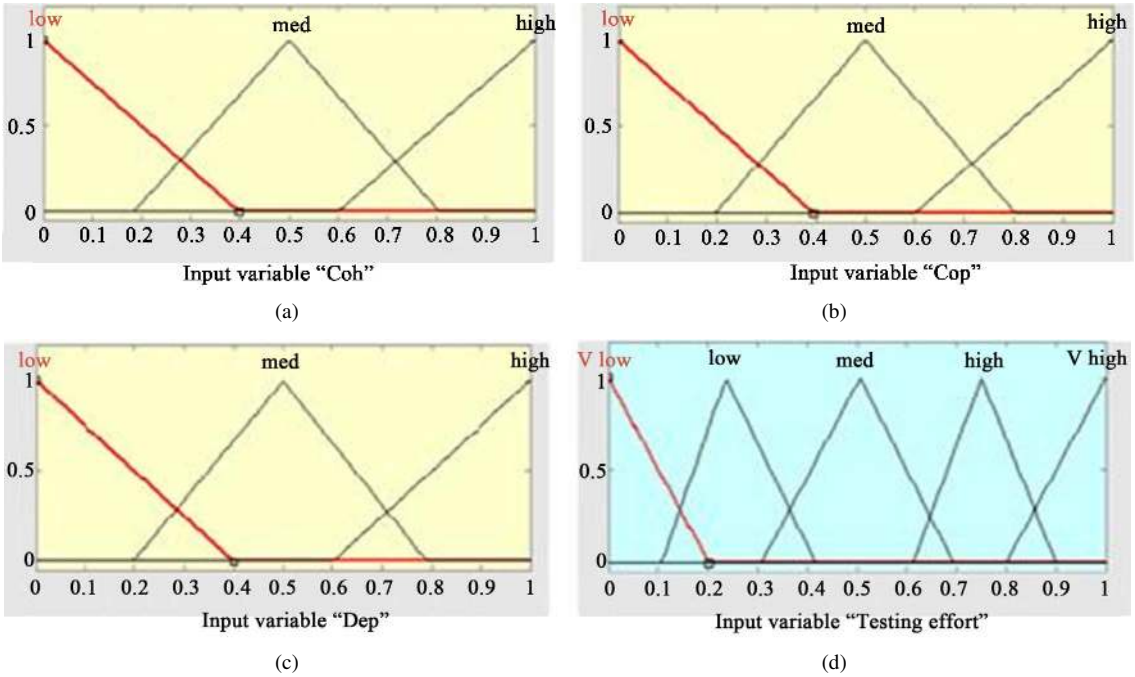


Figure 4. Membership function for (a) Coh; (b) Cop; (c) Dep input variables; and (d) testing effort output variable.

is illustrated in **Figure 4(c)**. The range of membership functions for all input variables is between 0 and 1. The membership functions are uniformly distributed over the range. The output from the fuzzy inference is the testing effort. This variable has five values; very low, low, medium, high, and very high. The membership functions for these values are illustrated in **Figure 4(d)**. The higher the value of this variable, the worse the testing effort required to test the system/component. As for input variables, the membership functions are uniformly distributed over the range 0 to 1.

Proposed Model Phases

The proposed model consists of two phases; system assessment and components assessment. In this section these phases are discussed in details.

- ***Phase One*: *System Assessment***

In this phase, the system architecture is assessed based on SCoh, SCop, and CCD. These metrics are calculated using equations 2, 6, and 5 respectively. The calculated values are entered to the fuzzy inference as Coh, Cop, and Dep respectively. The output from the fuzzy inference is the effort required for system testing. If this value is significant, phase two can be performed to reduce this value as much as possible.

- ***Phase Two*: *Components Assessment***

The purpose of this phase is to classify system components into different classes according to its testing efforts. This will direct the architect to the group of components that require a high testing effort. Then, he/she can modify them to decrease their required testing effort which means increasing the system testability. To assess components testability, component CCoh, CCop, and CDep are calculated according to Equations (1), (3), and (4) respectively. These values are evaluated using the fuzzy inference to know the testing efforts for that component. This process is repeated for each component. At the end of this phase, components are classified according to their testing efforts.

6. Case Study: Elders Monitoring System

The model has been tested using fall detection-response subsystem from Helping Our People Easily (HOPE) [16] [17]. The subsystem works as follows: Once a fall is detected, the network availability (cellular or Wi-Fi) is checked. For displaying user location in map view, the location co-ordinates are scanned and sent to one of map services using a Java Script Object Notation (JSON) query to retrieve and detect the location. If there is no net-

work available, then a recently cached location is used and GPS-based positioning scanning is stopped to prevent battery drainage.

For sending out an alert, the value of network status is checked. If cell network is available, then a call is dialed. In case there is no cell network but Wi-Fi network is available, then a message is sent out with the user location. For the worst case scenario of no networks being available, the alarm signals for help nearby. The architecture design is shown in **Figure 5**.

Implementation and Results Discussion

The model was implemented using Matlab software and applied on the elders monitoring system architecture. As illustrated in **Figure 5**, the architecture consists of seven components; four of them have several modules. When applying the proposed model, firstly, phase one will evaluate the testability of the entire system. Since system testing effort that results from phase one is moderate, phase two will evaluate the testability of each component.

Table 2 lists the evaluation results for elders monitoring system architecture. It lists the cohesion and the coupling values for each phase. In addition, it shows the testing effort (TE) for each component that result from the fuzzy inference system. Notice that, three components out of seven have high testing effort (the highlighted components: *network scanner*, *power source* and *alert transmitter* component). As a result, the architect can modify these components in order to reduce entire system testing efforts. This is can be done by increasing the cohesion and decreasing the coupling of the components that required high testing efforts. As a result, the cohesion of alert trasmitter component is increased by connecting call dialer and message transmitter modules to alert raiser module.

By this modification, the coupling between call dialer and alert switcher, and message transmitter and alert switcher are eliminated. Thus, reducing the total coupling of alert transmitter component. Another modification, to increase the cohesion of power resource component is to separate the charger module into different component since it does not have any connection with battery module or any other external modules. **Figure 6** shows the architecture after the modifications. The architecture is assessed again to evaluate the modification results.

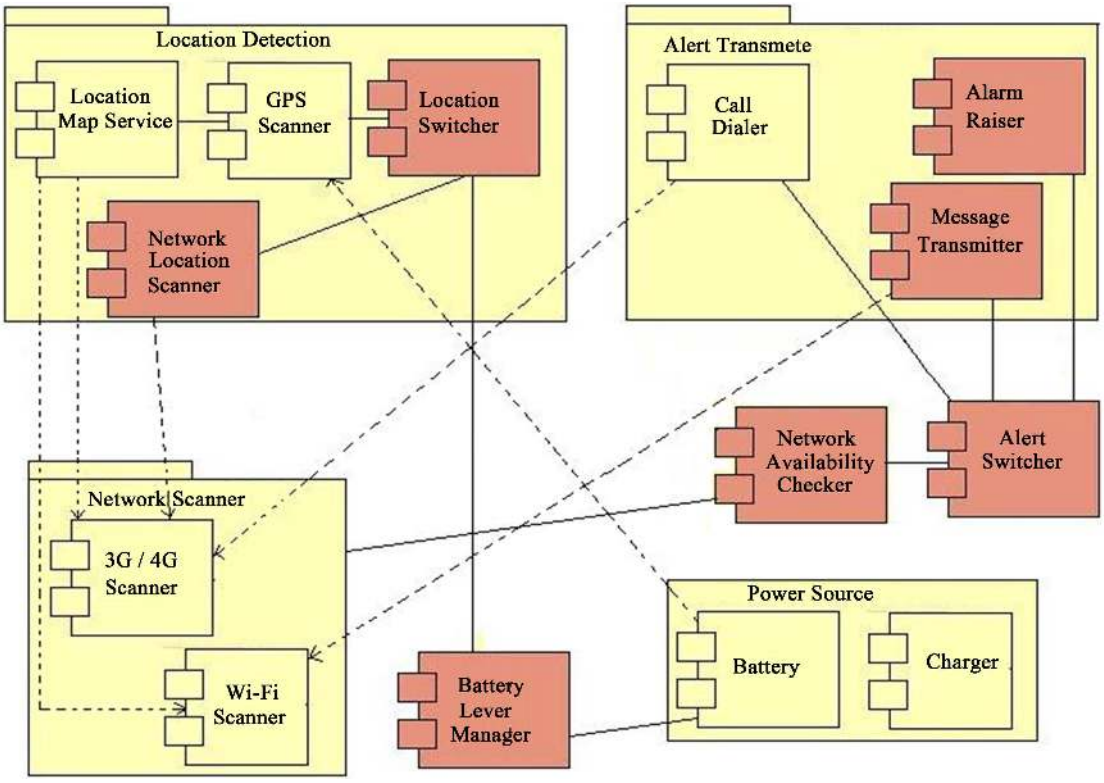


Figure 5. Elders monitoring system architecture.

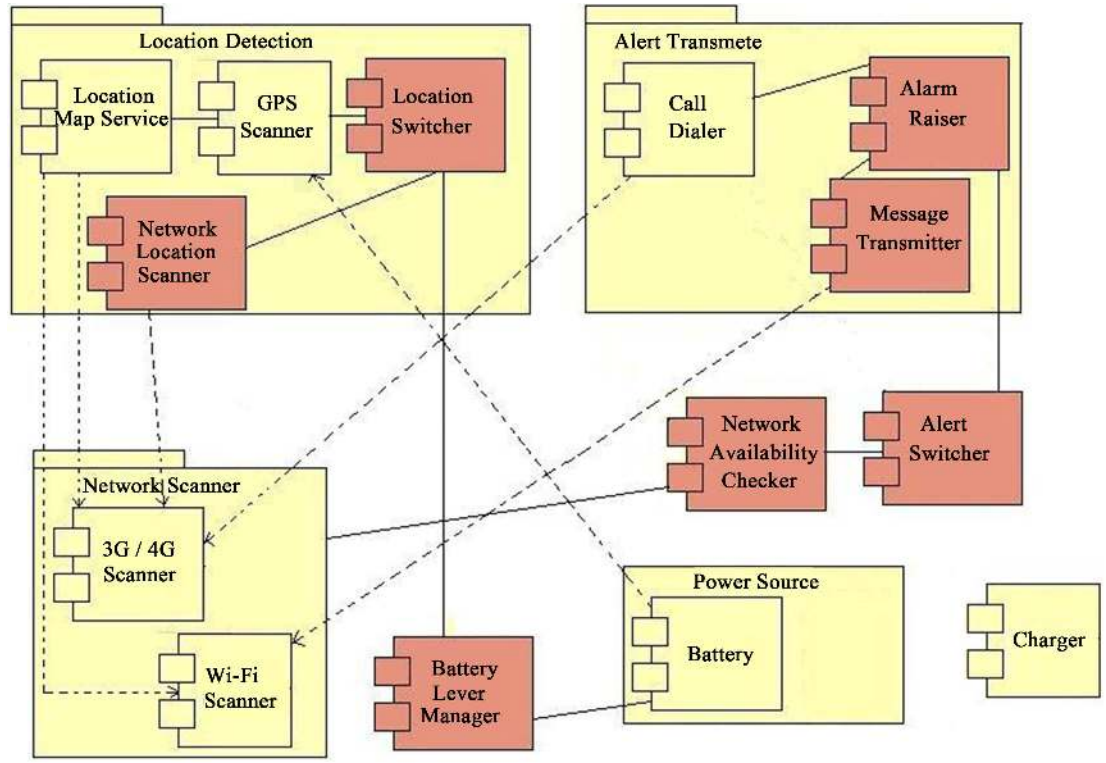


Figure 6. Elders monitoring system architecture after modifications.

Table 2. The evaluation results for elders monitoring system architecture before modification.

Phase 1	System	SCoh	SCop	CCD	TE	
		0.5	0.28	0.75	0.60	Moderate
Phase 2	Component	CCoh	CCop	CDep	TE*	
	Location detection	0.5	0.30	1.0	0.62	Moderate
	Network scanner	0.0	0.35	0.57	0.68	High
	Power source	0.0	0.15	1.0	0.75	High
	Alert transmitter	0.0	0.4	0.57	0.75	High
	Battery level manager	1.0	0.2	1.0	0.26	Low
	Network availability checker	1.0	0.2	0.57	0.09	Very Low
	Alert switcher	1.0	0.40	0.57	0.50	Moderate

The results are listed in **Table 3**. **Table 4** expresses the percentage values of the testing efforts improvement, where "-" means the decreasing of testing efforts.

Table 4 shows that the system testing effort is reduced by 42% since the testing effort for *power source* and *alert transmitter* is reduced by 67% and 35% respectively.

Figure 7 shows the comparison between system cohesion, coupling, dependency, and testing efforts before (called System_B) and after (called System_A) the architecture modifications. As illustrated in **Figure 7**, SCoh increases after architecture modifications. In addition, SCop and CDD is reduced by 14% and 21% respectively. These enhancements results in reducing system testing effort by 42%.

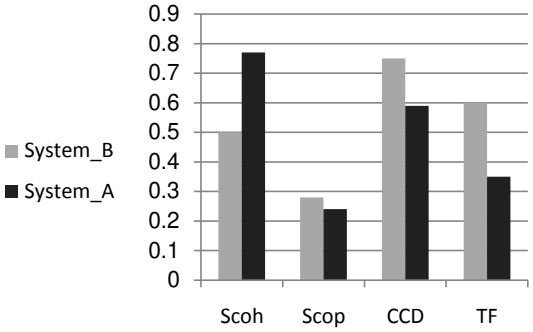


Figure 7. Comparison between system testability metrics before (System_B) and after (System_A) modification.

Table 3. The evaluation results for elders monitoring system architecture after modification.

Phase 1	System	SCoh	SCop	CCD	TE*	
		0.77	0.24	0.59	0.35	Low
Phase 2	Component	CCoh	CCop	Cdep	TE*	
	Location detection	0.50	0.37	0.87	0.71	High
	Network scanner	0.0	0.43	0.50	0.75	High
	Power source	1.0	0.18	0.87	0.25	Low
	Alert transmitter	0.66	0.25	0.50	0.49	Moderate
	Battery level manager	1.0	0.25	0.87	0.42	Moderate
	Network availability checker	1.0	0.25	0.50	0.29	Low
	Alert switcher	1.0	0.25	0.50	0.29	Low
	Charger	1.0	0.0	0.12	0.07	Very Low

Table 4. The percentage values of the testing efforts improvement. *Testing Effort, "-" means decreased.

Phase 1	System	SCoh	SCop	CCD	TE*
		54	−14	−21	−42
Phase 2	Location detection	CCoh	CCop	CDep	TE*
		0	23	−13	15
	Network scanner	0	23	−12	10
	Power source	100	20	−13	−67
	Alert transmitter	66	−38	−12	−35
	Battery level manager	0	25	−13	62
	Network availability checker	0	25	−12	222
	Alert switcher	0	−38	−12	−42
	Charger				

7. Conclusion and Future Work

In this paper, we proposed an evaluation model to assess software architecture (Architecture Design Testability Evaluation Model (ADTEM)) based on testability metrics. The model directs software architects on how to im-

prove software architecture testability. A fuzzy inference system was built to assess both software and component testability according to inference rules. The model was implemented and applied on a case study. The results showed that the model is applicable and efficient and it can improve the testability efforts. As future works, we propose an adjustment fuzzy rules and membership functions based on testable architectures.

References

[1] Dahiya, S., Bhutani, S., Oberoi, A. and Singh, M. (2012) A Fuzzy Model for Object Oriented Testability and Its Performance. *International Journal of Information and Technology and Knowledge Management*, **5**, 484-489.

[2] Sommerville, I. (2011) Software Engineering. 9th Edition, Pearson.

[3] Evaluating Software Architecture (2008) Software Architecture Book, Advanced Topics in Science and Technology in China, 221-273.

[4] Bruntink, M. (2003) Testability of Object-Oriented Systems: A Metrics-Based Approach. Master Thesis, Universiteit van Amsterdam, Amsterdam.

[5] Baudry, B., Traon, Y.L. and Sunyé, G. (2002) Testability Analysis of a UML Class Diagram. *Proceedings of the* 8*th IEEE Symposium on Software Metrics*.

[6] Badri, L., Badri, M. and Tour, F. (2011) An Empirical Analysis of Lack of Cohesion Metrics for Predicting Testability of Classes. *International Journal of Software Engineering and Its Applications*, **5**, 69-85.

[7] Badri, M. and Toure, F. (2012) Empirical Analysis of Object-Oriented Design Metrics for Predicting Unit Testing Effort of Classes. *Journal of Software Engineering and Applications*, **5**, 513-526.

[8] Chidamber, S.R. and Kemerer, C.F. (1994) A Metrics Suite for Object Oriented Design. *IEEE Transactions on Software Engineering*, **20**, 476-493. http://dx.doi.org/10.1109/32.295895

[9] Magiel, B. and Deursen, A.V. (2004) Predicting Class Testability Using Object-Oriented Metrics. *Proceedings of the* 4*th IEEE International Workshop on Source Code Analysis and Manipulation*, 136-145.

[10] Singh, Y. and Saha, A. (2008) A Metric-Based Approach to Assess Class Testability. *Agile Processes in Software Engineering and Extreme Programming Lecture Notes in Business Information Processing*, **9**, 224-225. http://dx.doi.org/10.1007/978-3-540-68255-4_30

[11] Gupta, V., Aggarwal, K. and Singh, Y. (2005) A Fuzzy Approach for Integrated Measure of Object-Oriented Software Testability. *Journal of Computer Science*, *Science Publications*, **1**, 276-282.

[12] Ahuja, H.K. and Kumar, R. (2012) Fuzzy Logic Driven Testability Measurement for an Object Oriented System. http://dspace.thapar.edu:8080/dspace/handle/10266/2064?mode=full&submit_simple=Show+full+item+record

[13] Kaur, N. (2011) A Fuzzy Logic Approach to Measure the Precise Testability Index of Software. *International Journal of Engineering Science and Technology*, **3**, 857-865.

[14] Dias, O.P. (1999) Metrics and Criteria for Quality Assessment of Testable Hw/Sw Systems Architectures. *Journal of Electronic Testing*: *Theory and Applications*, **14**, 149-158. http://dx.doi.org/10.1023/A:1008374027849

[15] Cumulative Component Dependency (CCD) (2013) http://baruzzo.wordpress.com/2009/08/22/how-testable-is-a-software-architecture/

[16] Helping Our People Easily (HOPE) (2013) http://www.utdallas.edu/~rym071000

[17] Chung, L., Lim, S., Chung, Y., Mehta, R.Y. and Chembra, A.B. (2010) AAC for Elderly People with Hearing, Speech or Memory Loss Due to Aging. Proceedings of the 25*th Annual International Technology & Persons with Disabilities Conference*, San Diego.