
Azam Farrokh a, Seyed Morteza Babamir a,∗

a Department of Computer Engineering, University of Kashan, Kashan, Iran.

ARTICLE INFO.

Article history:
Received: 29 August 2014
Revised: 7 December 2014
Accepted: 5 November 2015
Published Online: 7 February 2016

Keywords:

ABSTRACT

In this article, our aim is to quantify the impact of three reference architectural styles, i.e. Aspect-peer-to-peer, Aggregator-escalator-peer, and Chain-of-configurators on the maintainability attribute in self-healing systems. To do so, we used metrics of coupling and cohesion. To show the effectiveness of our method, we applied it to the Wireless Body Area Network (WBAN) as a case study. Our findings showed the Chain-of-configurators architectural style, in comparison with two other styles, had the least (best) coupling and the most (best) cohesion. In fact, it was the best style from the maintainability point of view.

1 Introduction

Self-adaptive systems are systems that are able to adapt themselves to the new conditions of their environment through learning from their past experiences. A self-healing system is a self-adaptive system which recovers from unexpected and impairing events [1, 2]. In designing the software of self-healing systems, architectural styles play important roles. [3] defined a style as: ”A specialization of element and relationship types, together with constraints on how they may be used.” In fact, an architectural style provides an abstraction of a family of systems. Accordingly, a suitable architectural style is a contributory factor for satisfying quality attributes of a system.

Among others, three reference architectural styles, i.e., Aspect-peer-to-peer, Aggregator-escalator-peer, and Chain-of-configurators, have been presented and discussed for designing software of self-healing systems [4]. The first style assigns a monitor component and configurator to each condition or aspect of the system. In the second style, the monitors are grouped and each group is allocated to one condition or aspect of the system and in the third style, several configurators could be chained together.

Architectural styles, such as maintainability, have a considerable influence on quality attributes of software. Therefore, quantifying the impact of a style on quality attributes helps software developers in selection of the best style. For self-healing systems, maximizing the maintainability attribute was considered as one of the main objectives. [5] defined the maintainability attribute as: “The degree of effectiveness and efficiency with which a product or system can be modified by the intended maintainers.” Two effective software engineering principles that are taken into account to satisfy quality attributes by a software system are coupling and cohesion; accordingly, we used these principles to quantify the impact of the three reference architectural styles on the maintainability of self-healing systems. Moreover, we proposed a number of metrics to measure coupling and cohesion of the three reference software architectural styles. Afterwards, we dealt with quantifying the impact of styles on the Wireless Body Area Networks (WBAN) system as a case study.
The rest of the paper is organized as follows. In Section 2, the related work is discussed and in Section 3, some properties of self-healing systems are introduced. In Section 4, the software architecture is described and in Section 5, the effective metrics for quantifying the impact of software architectural styles on the satisfaction of the maintainability attribute is explained. In Section 6, the WBAN system is proposed as a case study to show the effectiveness of our method. Finally in Section 7, conclusions are drawn.

2 Related Works

In this study, we consider the quantitative evaluation of the impact of software architectural styles on the maintainability of self-healing systems. Accordingly, we consider first, the studies that have addressed the quantitative evaluation of software architectural styles’ influence on quality attributes and, second, the studies that dealt with the quantitative evaluation of the maintainability attribute.

Software architecture evaluation methods are based on: (1) scenario, (2) simulation (3) mathematics and (4) measurement [6, 7]. The measurement-based evaluation methods use metrics for the evaluation, so they are more accurate than the others. Although, [8] discussed qualitative evaluations, these methods are less accurate than the quantitative and measurement-based methods. Therefore we consider a quantitative method for evaluation of quality attributes. Our quantitative method uses coupling and cohesion metrics to quantitatively measure maintainability by three reference software architectural styles applied to self-healing systems.

Some software attributes have already been evaluated by quantitative methods such as Markov models. [9] introduced a Markov model to utilize software architectural styles for assessing the reliability and performance attributes. Although they used a quantitative method but in contrast to our method, they did: (1) not use the software engineering metrics, coupling and cohesion, (2) not address the self-healing systems and (3) not quantitatively measure the maintainability.

For the software architecture of self-healing systems, three reference styles were introduced by [4] based on which [8] developed an analysis and reasoning framework to evaluate the modifiability attribute of the software architecture. However, they tailored a non-quantitative method for their work. Compared to this method, our work is different in two aspects: (1) we propose a quantitative method and (2) our method measures the maintainability with the reference styles. The similarity of our works is the use of reference styles as the basis.

As stated in Section 1, for self-healing systems, maximizing the maintainability is one of the main objectives. To evaluate the maintainability in the reference styles, [10] presented a method, which is similar to our method in two aspects: (1) considering the reference styles as the underlying principles and (2) assessing the level of maintainability. However, their method exploited a non-quantitative method which means they have not quantitatively measured the maintainability attribute in the reference styles.

3 Self-healing Systems

Soaring exchange of information among software systems, and their complexity necessitates the long life and high availability of such systems. To this end, they should be self-adaptive meaning that they should be able to adapt themselves to their environment conditions at run-time. Such systems are able to learn from their past experiences to predict some future condition.

Various definitions of self-adaptive software have been presented. Self-adaptive software is defined as a closed-loop system with a feedback loop [11]. Developing self-adaptive software through an adaptation engine and feedback loops, was proposed by [12]. The engine provides the adaptation at runtime through adaptable software models.

To satisfy users’ requirements, self-adaptive systems should respond to changes of their environment at run-time. Therefore, a self-adaptive system should: (1) monitor its environment changes and (2) react to the changes [13]. To this end, it needs to specify the critical goal that should be satisfied. Furthermore, since the self-adaptive systems are expected to have specific properties, they are called systems with self-* properties [14, 15]. For the adaptation process, the following questions should be answered [13]:

(1) Which parts of the system should be adapted?
(2) Which aspects of the system should be changed or maintained at run-time?

From the control point of view, one technique for implementing a self-adaptive system is adding a separate and external control unit to monitor and adapt the system during the run-time [13]Cheng; however, this method is expensive. Another method is monitoring and adapting a system at run-time based on the software architectural model of the system [16]. Such methods are called architecture-based self-adaptation.

Self-adaptive systems are classified into four self-* groups: self-configurable, self-optimized, self-protectable and self-healing [1, 17]. Self-healing systems are able to diagnosis bugs, errors, and the cause
of failure; furthermore, they can recover from an unexpected event [1]. In this article, we concentrate on this type of self-adaptive systems.

4 Software Architecture

The structural design of a software system is a critical aspect of software design because it provides a high-level view of software components and their communication. [5] states: “The software architecture of a system is the set of structures needed to reason about the system, which comprise software elements, relations among them, and properties of both.”

The rest of this section is organized as follows. In Section 4.1, the relationship between software architecture and quality attributes is discussed. Sections 4.2 and 4.3 are dedicated to software quality attributes and the concept of coupling and cohesion respectively.

4.1 Software Architecture and Quality Attributes

Satisfying quality attributes of a system is largely determined by its architecture; therefore, understanding the relationship between the software architecture and quality attributes is a significant concern. Designing a system without supporting quality attributes may lead to the redesign of the system during its lifetime. If we are able to evaluate a system before designing, we can avoid its redesign. Having carried out such an evaluation, we will be able to select the most suitable software architecture based on its level of support for quality attributes.

To evaluate the software architecture of a system, different compositions of the software components should be evaluated. A specific composition of components with a number of constraints governing the composition is called an architectural style, which is a building block of the software architecture. Since a style includes the features influencing quality attributes, we can evaluate the style by identifying its degree of support for quality attributes. Architectural styles have a strict impression on system quality attributes such as reliability and maintainability. This is why selecting an appropriate architectural style is an important step for development of software [17].

4.2 Software Quality Attributes

In this section, we explain software quality attributes. In software engineering, software with highly satisfied quality attributes means there is no end users’ dissatisfaction problem. There is a standard for the evaluation of software quality [18] whose first part consists of the classified software quality attributes, Functionality, Reliability, Usability, Efficiency, Maintainability and Portability. In addition, the self-healing systems have more features due to their changes and adaptations at runtime.

Self-healing is the ability of detection and recovery from a failure and the main purpose of developing such a system is the concentration on quality attributes such as availability, durability, maintainability and reliability [2]. The relation between attributes of self-adaptive systems and common quality attributes was surveyed by [19]. As stated in Section 1, among others, maximizing the maintainability attribute is one of the main objectives of self-healing systems. In Section 1, the definition of the maintainability attribute by Bass et al was presented. ISO 9126 knows maintainability as one of the six major features of the software product quality where adaptability, changeability, stability, and testability are counted as sub-features of the maintainability [18]. The software maintainability attribute depends on the software design modularity, i.e., a design with low coupling and high cohesion of modules. In other words, the sub-features of the maintainability attribute of software will be improved when the coupling/cohesion degree between the software modules decreases/increases. The impact of cohesion and coupling on software maintainability is emphasized in [20]. We explain cohesion and coupling more in the following section.

4.3 Cohesion and Coupling

Generally speaking, coupling is an attribute of a pair of modules indicating the degree of interdependency between them [21]. Based on the experience with design of some software systems, [22] presented the numeric values 0 to 5 for types of couplings (Table 1).

According to [21] the coupling types are:

(1) No coupling: modules do not communicate with each other; this means that they are totally independent of each other,

(2) Data coupling: two modules share data with each

<table>
<thead>
<tr>
<th>Coupling type</th>
<th>Weight</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Coupling</td>
<td>0</td>
<td>w0</td>
</tr>
<tr>
<td>Data</td>
<td>1</td>
<td>w1</td>
</tr>
<tr>
<td>Stamp</td>
<td>2</td>
<td>w2</td>
</tr>
<tr>
<td>Control</td>
<td>3</td>
<td>w3</td>
</tr>
<tr>
<td>Common</td>
<td>4</td>
<td>w4</td>
</tr>
<tr>
<td>Content</td>
<td>5</td>
<td>w5</td>
</tr>
</tbody>
</table>
other by the parameter passing mechanism,
(3) Stamp coupling: two modules share a composite 
data structure by parameter passing mechanism,
(4) Control coupling: a module controls the process 
of another by a control flag passing,
(5) Common coupling: some modules share the pro-
gram global data,
(6) Content coupling: a module can change the in-
ternal workflow of another.

[21] defined module cohesion as, how a module 
needs the other components to perform a job. They 
introduced the types of cohesion as follows. In this 
article, we use the numeric values 0 to 6 for types of 
cohesion (Table 2).

<table>
<thead>
<tr>
<th>Cohesion Type</th>
<th>Weight</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coincidental</td>
<td>0</td>
<td>w0</td>
</tr>
<tr>
<td>Logical</td>
<td>1</td>
<td>c1</td>
</tr>
<tr>
<td>Temporal</td>
<td>2</td>
<td>c2</td>
</tr>
<tr>
<td>Procedural</td>
<td>3</td>
<td>c3</td>
</tr>
<tr>
<td>Communicational</td>
<td>4</td>
<td>c4</td>
</tr>
<tr>
<td>Sequential</td>
<td>5</td>
<td>c5</td>
</tr>
<tr>
<td>Functional</td>
<td>6</td>
<td>c6</td>
</tr>
</tbody>
</table>

4.4 Architectural Styles for Self-healing Systems

According to the style definition by Clements et al. (see 
Section 1), a style introduces an architectural structure 
consisting of elements such as modules, components, 
and connectors through recognizing the elements and 
their interrelationship.

A self-healing system must be able to adapt its be-
behavior to the changes of its environment. To this end, it 
must be able to continuously monitor its environment 
and react accordingly. Therefore, an architecture of a 
self-healing system should regard three basic mecha-
isms [4]: (1) Reflection by which the system responds 
to its internal and external conditions, (2) Reasoning 
by which the system takes an action in response to a 
condition, and (3) Configuration by which the system 
-adapt itself in response to the conditions (Figure 1).

In the reference architectural styles, the compo-
nent runtime reflection and component configuration 
are called the monitor and configurator respectively. 
Now, we explain the reference architectural styles for 
self-healing software systems based on monitor and 
configurator components:

(1) Aspect-peer-to-peer style: This style (Fig-
ure 2) assigns a monitor component to each con-
dition or aspect of the system that should be 
monitored; furthermore, each monitor has a cor-
responding configurator to perform the needed 
adaptations.

(2) Aggregator-escalator-peer style: In this 
style, a monitor sends the observed data to high-
level aggregator monitors (Figure 3), which 
then pack the data to a composite package. Then, 
the composite package is passed to its 
respective configurator.

(3) Chain-of-configurators style: In this style, 
several configurator components are chained 
together. The configuration request is moved 
along with the chain so that it can be successfully 
processed. Figure 4 shows this style consisting 
of a single monitor in the entire system; if one of 
the configurators is not able to perform a request 
successfully, one of the other configurators will 
try to do the request. Moreover, configurators 
can perform tasks together by dividing the task
among themselves [10].

![Diagram of Aspect Peer-to-Peer Architectural Style](image)

**Figure 2.** Aspect Peer-to-Peer Architectural Style [4]

![Diagram of Aggregator-Escalator-Peer Architectural Style](image)

**Figure 3.** Aggregator-Escalator-Peer Architectural Style [4]

![Diagram of Chain-of-Configurators Architectural Style](image)

**Figure 4.** Chain-of-Configurators Architectural Style [4]

5 Quantitative Measurement of Architectural Styles for Self-healing Systems

As explained in Section 4.1, since quality attributes, such as reliability and maintainability are influenced by an architectural style, selecting an appropriate style becomes a significant concern in designing software and satisfying users’ requirements [17]. On the other hand, the main purpose of developing a self-healing system as an adaptive system is meeting quality attributes such as availability, durability, maintainability and reliability of the system [2]. Among others, maintainability is influenced by software cohesion and coupling [20] meaning that by increase/decrease of coupling/cohesion between software components, maintainability will be reduced. In the following sections, we use the proposed coupling and cohesion metrics for measuring the satisfaction score of qualities gained by the reference styles (see Section 4.4).

5.1 The Coupling Measurement Method

As stated in Section 2, Hawthorne and Perry proposed three architectural styles for self-healing systems [4] but they didn’t propose any method for measuring the coupling and cohesion of those styles. In this article, we generalized the concept of “module coupling” to coupling of software architecture components and used it to measure the coupling degree of the software architectural styles for self-healing systems. We consider the relation between monitors and configurators (see the previous section) as a coupling relation because they are two separate components. The coupling of a style is calculated in regard to the type of coupling and the number of components that the style is composed of. The more the coupling between the style components increases, the more the understandability, correctness and maintainability of the style decrease [20].

We compute the coupling value of a style (consisting of monitors or configurators) as sum of coupling values of its components (indicated by $CPS$, Equation (1)). Similarly, the coupling value of a component is calculated as sum of its couplings to other components (Equation (2), where notation $k$ denotes the number of couplings between component $i$ and other components, and notation $w_j$ indicates the weight of coupling between components $i$ and $j$, indicated by $CP_j$, see Table 1 in Section 4.3. By increasing the style’s coupling (i.e. $CPS$ in Equation (1) of components, the maintainability will reduce [20]; this means the coupling and maintainability are inversely related. Measuring coupling of software architectural styles for self-healing systems is explained in Section 5.3.

\[
CPS = \sum_{i=1}^{n} CPC_i \quad \quad (1)
\]

\[
CPC_i = \sum_{j=1}^{k} w_j \ast CP_j \quad \quad (2)
\]
5.2 The Cohesion Measurement Method

In this article, we generalized the concept of “module cohesion” for the cohesion of software components and used it to measure the cohesion values of software architectural styles. For example, jointly performing a special task by two or more consolidated component modules can be considered as cohesion. The more the cohesion of components increases, the more the understandability, modifiability, and maintainability of the components increase. Similar to Equations (1) and (2), the cohesion of a style (indicated by \( CHS \)) and its components (indicated by \( CHC \)) are computed as Equations (3) and (4) (see Table 2).

\[
CHS = \sum_{i=1}^{n} CHC_i \quad (3)
\]

\[
CHC_i = \sum_{j=1}^{k} w_j \cdot CH_j \quad (4)
\]

By increasing the cohesion between the components, the software maintainability will increase. In fact, the cohesion and maintainability are directly related together. The cohesion measurement of software architectural styles for self-healing systems is explained in Section 5.4.

5.3 Measuring the Style Coupling

(1) **Aspect-peer-to-peer style:** This architectural style thinks of the software as a set of independent elements. The monitor and its respective configurator belong to an element and there is no relation between elements. Thus, such a strict peer-to-peer approach decreases coupling. It can be useful when configurators can decide based on just their respective monitors and not monitors of other components. This means that the configurator’s need for the information provided by other monitors leads to Data coupling (See Table 1). Moreover, since monitors may send control flags to configurators, the control coupling may occur between monitors and configurators. Therefore in this style, there is no coupling of type 1 or 2 (Data or Control). Given that there are \( n_1 \) couplings (consisting of \( n_{11} \) Data and \( n_{12} \) Control couplings) between a component and other components, the style coupling is computed as Equation (5) (see Equations (1) and (2) and Table 1). Note that coupling between two components is taken into account just one time.

\[
\sum_{i=1}^{n_1} CPC_i = \sum_{i=1}^{n_1} w_1 + \sum_{i=1}^{n_1} w_2 = n_{11}w_1 + n_{12}w_2 \quad (5)
\]

(2) **Aggregator-escalator-peer style:** This architectural style contains a set of independent elements arranged in a hierarchical structure. Each monitor component sends data to its upper-level monitor, which indicates a Data Coupling. The upper-level monitors pack data received from the lower-level and pass them to a peer configurator, which indicates a Stamp Coupling. Moreover, each upper-level monitor sends a control flag to its respective configurator, which denotes a Control Coupling. Configurators use the data received from the upper-level configurators, which denotes a Common Coupling. Therefore, between each two components of this style there exist at most 4 coupling types (Data, Stamp, Control, and Common). Given that there are \( n_2 \) couplings (consisting of \( n_{21} \) Data, \( n_{22} \) Sample, \( n_{22} \) Control, and \( n_{21} \) Common couplings) between two components, the style coupling is computed as Equation (6) (see Equations (1) and (2) and Table 1). Note that coupling between each two components is taken into account just once.

\[
\sum_{i=1}^{n_2} CPC_i = \sum_{i=1}^{n_2} w_1 + \sum_{i=1}^{n_2} w_3 + \sum_{i=1}^{n_2} w_3 + \sum_{i=1}^{n_2} w_3 = n_{21}w_1 + n_{22}w_2 + n_{22}w_3 + n_{21}w_4 \quad (6)
\]

(3) **Chain-of-configurators style:** In this style, configurators collaborate to carry out tasks where each configurator passes data to its adjacent after completing its task. The process continues until performing all tasks successfully. [10]. Therefore, there is a Data Coupling between each two configurators except for the last one, which has no more configurators to send the data to. This is why there exists just one coupling between each two components of this style. Accordingly, for an architecture consisting of \( n_3 \) components, the total style coupling is computed as Equation (7).

\[
\sum_{i=1}^{n_3} CPC_i = \sum_{i=1}^{n_3} w_1 = (n_3 - 1)w_1 \quad (7)
\]

Below, the coupling equations for the three reference styles are shown.

Aspect-peer-to-peer:
5.4 Measuring Style Cohesions

(1) **Aspect-peer-to-peer style:** In this style, each element includes a monitor and its respective configurator; therefore, there isn’t any type of cohesion between them.

(2) **Aggregator-escalator-peer style:** In this style for each subsystem of the software there are two monitors, upper-level and lower-level, which they are related to a special aspect of the system. Each lower-level monitor supervises one part of the subsystem and the monitors logically related to. This means that there is Logical cohesion between monitors. Each configurator receives data from its upper-level ones, which denotes the existence of Communicational cohesion respectively. Note that cohesion between each two configurators is taken into account just once.

\[
\sum_{i=1}^{n} CPC_i = \sum_{i=1}^{n1} w_1 + \sum_{i=1}^{n2} w_2 = n_{11} w_1 + n_{12} w_2
\]

Aggregator-escalator-peer:

\[
\sum_{i=1}^{n2} CPC_i = \sum_{i=1}^{n21} w_1 + \sum_{i=1}^{n22} w_3 + \sum_{i=1}^{n22} w_3 + \sum_{i=1}^{n21} w_4
\]

Chain-of-configurators:

\[
\sum_{i=1}^{n3} CPC_i = \sum_{i=1}^{n3} w_1 = (n_3 - 1) w_1
\]

Below the cohesion equations for three reference architectural styles are shown.

**Aspect-peer-to-peer:** Zero

**Aggregator-escalator-peer:** \(m_{21} c_1 + m_{22} c_4\)

**Chain-of-configurators:** \((m_3 - 1) c_3 + (m_3 - 2) c_5\)

6 Case Study: Wireless Body Area Network

Nowadays, in many countries, the cost of health care is rising sharply. However, the main cause of more than 30% of deaths is the cardiovascular disease that most of them can be prevented by health care [23]. This has culminated in introducing the new systems that constantly monitor a patient. Among others, Wireless Body Area Networks (WBAN) is a system that constantly monitors medical parameters of the patient’s body and may be used with the patient in the hospital, at home or on the move.

A WBAN system contains various sensors implanted in the patient’s body or worn by the patient for receiving/sending data from/to the patient’s body. These sensors send the medical information to an external server such as a pocket PC lying outside of the patient’s body (for instance, in the patient’s clothing pocket). The server is responsible for storing, maintaining and analyzing patient’s body data [24]. Since the wired connections between the server and the patient are difficult and expensive to handle, wireless connections are used. WBANs have the ability to self-heal patients leading to mitigation of the need for continuous surveillance by the physician.

6.1 Sensors and Actuators

Generally speaking, a WBAN system consists of Sensors and Actuators [25] (Figure 5), where sensors measure the patient’s medical parameters such as, glucose level, heartbeat, body temperature or heart electrical changes and actuators are responsible for performing suitable actions on the patient’s body according to the data received from the sensors or through
6.2. Design of WBAN Using Aspect-Peer-to-Peer Style

Table 3 shows the design of the WBAN system using the aspect-peer-to-peer architectural style where: (1) sensors have been intended for: Electroencephalogram (EEG), auditory, motion, position, Electrocardiogram (ECG), heart rate, blood, fingertip, blood pressure, and blood oxygen and (2) actuators have been included in hearing aid cochlear implant, artificial knee, artificial arm, blood pump, insulin injection, and pacemaker.
Table 3. Design of WBAN Using the Aspect-Peer-to-Peer Style

<table>
<thead>
<tr>
<th>S#</th>
<th>Sensor Name (Connected Actuators)</th>
<th>A#</th>
<th>Actuator Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>EEG (A2, A3)</td>
<td>A1</td>
<td>Hearing Aid</td>
</tr>
<tr>
<td>S2</td>
<td>Audiometry (A1)</td>
<td>A2</td>
<td>Artificial Knee</td>
</tr>
<tr>
<td>S3</td>
<td>Motion(A2, A3, A5, A6)</td>
<td>A3</td>
<td>Artificial Arm</td>
</tr>
<tr>
<td>S4</td>
<td>Positioning(A2, A3, A5, A6)</td>
<td>A4</td>
<td>Blood Pump</td>
</tr>
<tr>
<td>S5</td>
<td>ECG (A4)</td>
<td>A5</td>
<td>Insulin injection</td>
</tr>
<tr>
<td>S6</td>
<td>Heartbeat(A4, A8)</td>
<td>A6</td>
<td>Injection blood pressure increaser/ decreaser</td>
</tr>
<tr>
<td>S7</td>
<td>Blood (A5)</td>
<td>A7</td>
<td>Blood Oxygen</td>
</tr>
<tr>
<td>S8</td>
<td>Fingertip (A5)</td>
<td>A8</td>
<td>Pacemaker</td>
</tr>
<tr>
<td>S9</td>
<td>Blood Pressure(A6)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>S10</td>
<td>Oxygen (A7)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

are obtained for other actuators. Taking into account the data and control couplings mentioned above and based on Table 1, the coupling value of the aspect-peer-to-peer architectural style is calculated as:

\[ CPS = w_1(2 + 2 + 1 + 3 + 2) + w_3(1 + 3 + 3 + 1) = 1 \times 10 + 3 \times 8 = 34 \]

As explained in Section 5.4, the aspect-peer-to-peer architectural style has no cohesion; therefore, the cohesion value for this style is zero.

6.3 Design of WBAN Using Aggregator-Escalator Style

Figure 6 shows the design of WBAN using the aggregator-escalator-peer architectural style.

6.3.1 Identifying the System Components and Measuring Coupling and Cohesion

Figure 6 shows the WBAN configuration according to the aggregator-escalator style where individual monitors (sensors) and configurators (actuators) are aggregated to 4 complex monitors and 4 complex configurators respectively. As Figure 6 shows, 8 of 10 WBAN’s sensors constitute two separate complex monitors called ”motion” (including 3 individual sensors) and ”heart and blood” (including 7 individual sensors). These two complex sensors contain some common sensors such as motion and positioning; and 2 individual sensors, ”oxygen detection” and ”audiometry”, each one by itself constitute a separate monitor. Accordingly, there exist four monitors and similarly, four configurators in this style.

We measure the coupling value of the aggregator-escalator-peer architectural style for the WBAN system according to Equation (6). Each individual monitor sends its data to its respective complex monitor; therefore, there is a data coupling whose value is equal to the number of the individual monitors. Furthermore, each complex monitor collects data from its individuals and sends them to the corresponding actuator; this represents the stamp coupling. Moreover, each complex monitor sends a control flag to the corresponding actuator for announcing a new event; this indicates the control coupling, and therefore the total control coupling is equal to the number of complex monitors. Similarly, the individual actuators use the complex ones’ data, which indicates the common coupling.

According to couplings mentioned above, the coupling value of the aggregator-escalator-peer style for the WBAN system is calculated as:

\[ CPS = 10w_1 + 2(w_2 + w_3) + 6w_4 = 10 + 10 + 24 = 44 \]

The cohesion value of the aggregator-escalator-peer architectural style for the WBAN system is calculated according to Equation (8). Each individual monitor controls a particular aspect of system. This means that the individual monitors logically are connected to their respective complex monitors; therefore, there is a logical cohesion whose value is equal to the number of individual monitors. On the other hand, all individual actuators use data of their respective actuators; this means that there is the communicational cohesion whose value is equal to the number of individual actuators. Considering the cohesions mentioned above, the aggregator-escalator-peer style cohesion is calculated as:
6.4 Design of WBAN Using Chain-of-Configurators Style

As Figure 7 shows, there are a sensor (monitor) and seven actuators. In this style, actuators are chained together in the form of a linear structure where data are flowed among actuators. This denotes data coupling (see Equation (8)) whose value is calculated as:

\[
CPS = (n - 1) w_1 = (8 - 1) w_1 = 7 \times 1 = 7
\]

The cohesion value of the chain-of-configurators style for the WBAN is calculated according to Equation (9). In this style, a sequence of operations is flowed through actuators; this indicates there is procedural cohesion for each actuator except the last one. Furthermore, we have sequential cohesion for each actuator except the last one because the output of each actuator is used as input to its adjacent one. Since the WBAN system is composed of 8 actuators, the cohesion value of the chain-of-configurators style is calculated as:

\[
CHS = (8 - 1)c_3 + (8 - 1)c_5 \\
= 7 \times 3 + 7 \times 5 = 21 + 35 = 56
\]

Figure 8 shows the coupling and cohesion values of the three different reference styles for the design of the WBAN system. As Figure 8 shows, the chain-of-configurators style has the lowest coupling/cohesion value in comparison with other ones; consequently, it is the best style from the maintainability point of view.

7 Conclusions

In this article, we first found the significance of coupling and cohesion in assessing the support of maintainability quality attribute by software architectural styles. The support assessment was quantified by
proposing metrics for measuring the coupling and cohesion values. To show the significance of the metrics, we applied them to a self-healing system called the Wireless Body Area Network. Our second finding was the way of using the reference software architectural styles for the design of self-healing systems so that one can evaluate the design using coupling and cohesion criteria. It was shown by applying the method to WBAN self-healing system. We showed the comparison mechanism that one can establish for types of system designs obtained from software architectural styles for self-healing systems. Through the proposed mechanism, we concluded the best style from the coupling and cohesion points of view is the chain-of-configurators style for the WBAN system, when our concern is the system maintenance on account of the lowest coupling and the most cohesion values.

Acknowledgements
Authors like to thank University of Kashan for supporting this study by Grant No. 92/6188.

References
Azam Farrokh received a Bachelor of Science degree in software engineering from the University of Qom, Qom, Iran, 2010 with the thesis of “Simulation of nanoscale electrical discharge using MATLAB”. She received a Master of Science degree in Software Engineering from the University of Kashan, Kashan, Iran, 2012 with the thesis of “Quality evaluation of software architecture styles using architectural tactics”. She has already authored 6 international and internal conference papers.

Seyed Morteza Babamir received BS degree in Software Engineering from Ferdowsi University of Mashhad and MS and PhD degrees in Software Engineering from Tarbiat Modares University in 2002 and 2007 respectively. He was a researcher at Iran Aircraft Industries, Tehran, Iran, from 1987 to 1993, head of Computer Center in University of Kashan, Kashan, Iran, from 1997 to 1999 and head of Computer Engineering Department in University of Kashan from 2002 to 2005. He is associate professor of Computer Engineering Department in University of Kashan. He authored one book in Software Testing, 4 book chapters, 20 journal papers and more than 50 international and internal conference papers (http://ce.kashanu.ac.ir/babamir/Publication.htm). He is managing director of Soft Computing Journal published by supporting University of Kashan, Kashan, Iran.