Self-healing Architectural Styles: towards Quantitative Evaluation

Seyed Morteza Babamir* Azam Farokh

*Department of Computer Engineering, Faculty of Engineering, University of Kashan, Kashan, Iran.
Phone (Office): (+98)-361-5912469
Fax: (+98)-361-555-2930
Email: babamir@kashanu.ac.ir

Abstract In this article, we considered three reference architectural styles for self-healing systems, Aspect-peer-to-peer, Aggregator-escalator-peer, and Chain-of-configurators and dealt with quantifying them. To this end, we focused on the *maintainability* quality attribute and exploited *coupling* and *cohesion* principles. To employ these principles, we proposed metrics in order to quantify self-healing architectural styles. To show effectiveness of our method, we applied it to the Wireless Body Area Networks system as a case study. Our findings generally showed the Chain-of-configurators architectural style had the least coupling and the most cohesion amount in comparison with two other styles; so, it was the best style from the *maintainability* view.

Keywords: Self-adaptive system, Self-healing system, Architectural Style, Maintainability, Coupling, Cohesion.

1 Introduction

The systems that are able to adapt themselves to new conditions are called *self-adaptive* systems. For the adaption, they can learn from their recent past experiences. A specific type of self-adaptive systems is *self-healing* system (Kephart and Chess 2003; Ganek and Corbi 2003; Parashar and Hariri 2005) where the system is able to recover from unexpected events that may impair some parts of the system (Kephart and Chess 2003; Ganek and Corbi 2003; Parashar and Hariri 2005). In design of software for a self-healing system, architectural styles play an important role. Three are three reference architectural styles for design of such software. In this article, we focus on software architecture this type of self-adaptive systems. Software architecture includes software elements, the connection between them, and the properties governing the elements and connections. The software architecture also identifies the set of steps used to select, define, and design software architecture. An architectural style is a set of principles that provides an abstract framework for a family of systems. The selection of an appropriate architectural style has an important effect on system quality attributes.

In (Hawthorne and Perry 2005) several reference architectural styles for self-healing systems presented and discussed. These styles are included: 1) Aspect-peer-to-peer, 2) Aggregator-escalator-peer and 3) Chain-of-configurators. Aspect-peer-to-peer architectural style assigns a monitor component and a corresponding configurator to each condition or aspect of the system. In Aggregator-escalator-peer architectural style monitors are grouped and each group is allocated to one condition or aspect of the system. In Chain-of-configurators architectural style several configurator component are chained together.

Architectural style has a striking impression on system quality attributes such as reliability, maintainability. In other words, a quantitative impact of architectural style on quality attributes can help the software developers to select the best style. *Maintainability* is one of key quality attributes in self-adaptive systems, and particularly in self-healing systems (Mazeiar Salehie and Tahvildari 2005). *Coupling* and *cohesion* are two concepts that effect on maintainability; so, in this study we measured the coupling and cohesion of self-healing architectural styles to compare them. The metrics are essential in evaluation of software design quality, especially for comparing. As a result, in this paper we proposed metrics to measure coupling and cohesion of self-healing architectural styles. Then these formulas are discussed in Wireless Body Area Networks system as a case study.

The rest of the paper is organized as follows. In Section 2, some related works are discussed. Section 3 introduces the concepts of self-healing systems. Section 4 describes the software architecture. The metrics for quantitative measurements of self-healing architectural styles are given in Section 5. Finally, the application of these metrics in a case study is described in Section 6.

2 Related Work

Evaluating software architecture significantly reduces the cost. Until now, many methods have been proposed for evaluating software architecture and many of quality attributes of the software architecture are evaluated. According to the quality attributes evaluation methods in the software architecture, some methods for the quantitative evaluation of these features in software architecture styles has been proposed. In (Wang et al. 1999) and (Wang et al. 2006) the analytical model based on a discrete Markov model for evaluating the reliability of some architectural styles is proposed. Also the performance of software architectural styles is evaluated based on a discrete Markov model in (Sharafi et al. 2010).

By the development of self-healing systems, the need to evaluate these systems is also required. In (Neti and Muller 2007) an analysis and reasoning framework for self-healing systems is developed. This framework is based on self-healing architectural style proposed by Hawthorne and Perry in (Hawthorne and Perry 2005) and the analyze of modifiability attribute is considered. Some maintainability concerns in self-healing architecture is characterized in (Zhu et al. 2008).

Software architecture evaluation methods are divided into 4 groups as follows: scenario-based evaluation methods, simulation-based evaluation methods, mathematic model-based evaluation methods and measurement-based evaluation methods. Measurement-based evaluation methods uses metrics for evaluating and this method is very precise in comparing other methods. The evaluations discussed in (Neti and Muller 2007) and (Hawthorne and Perry 2005) are qualitative measurement. Qualitative evaluations are less accurate than quantitative and measurement-based assessment. In this article, we proposed a quantitative method for evaluating coupling and cohesion of self-healing architectural style.

3 Self-healing Systems

With the increasing information exchanging between software systems and the complexity of these systems, systems with long life and high availability are needed; so, we need to systems that can adapt themselves at run time. The systems that are able to adapt themselves are called Self-adaptive systems. These systems can change and improve themselves with respect to various conditions. Additionally these systems can use from their previous experience and they have learning ability.

Various definitions of self-adaptive software are provided in references. One definition is described in (Oreizy et al. 1999) is “Self-adaptive software modifies its own behavior in response to

changes in its operating environment. By operating environment, we mean any-thing observable by the software system, such as end-user input, external hardware devices and sensors, or program instrumentation." Another is provided in (Laddaga et al. 1997)" Self-adaptive software evaluates its own behavior and changes behavior when the evaluation indicates that it is not accomplishing what the software is intended to do, or when better functionality or performance is possible." Self-adaptive software means the use of self-adaptive concept in software domain. The life-cycle of self-adaptive software must be continued after its development (M. Salehie and Tahvildari 2009).

Managing the complexity of software systems to access their target have a rising costs, so this is basic requirement for self-adaptive software (Laddaga 2001). Self-adaptive software must respond to changes at run-time and also be able to satisfy some requirements (M. Salehie and Tahvildari 2009), so self-adaptive software is expected to have specific properties, that are called self-* properties (Kephart and Chess 2003; Babaoglu et al. 2005). Self-adaptive system should monitor the changes all the time and be able to react to changes (B. Cheng et al. 2009). But it is clear the system cannot monitor everything, so system needs to specify critical goal that should be satisfied in any condition (B. Cheng et al. 2009). So some questions that should be answered are (B. Cheng et al. 2009):

- What parts of the system need to be adapted?
- Which aspect of the system will change at run-time and which aspect must be maintained?

From the control view, one technique for implementing self-adaptive system is adding separate, external control unit that monitor the system during the run-time and adapt it (S. W. Cheng and Garlan 2007), but this method is expensive and not convenient. In (Dashofy et al. 2002 and Magee et al 1995; Oreizy et al. 1999) an architectural model of the software is suggested that it could be a useful basis for changing and monitoring the system at run-time. The architectural model that can be used as a foundation for monitoring and adapting a system at run-time is called **architecture-based self-adaptation** (S. W. Cheng and Garlan 2007)**.**

Self-adaptive systems are classified into four groups: self-configuring, self-optimizing, self-protecting systems and self-healing (Kephart and Chess 2003; Ganek and Corbi 2003; Parashar and Hariri 2005). Self-healing systems are able to diagnosis bugs, errors and cause of failure and they can recover from unexpected event (Kephart and Chess 2003). In this article, we focus on this type of self-adaptive systems. In (Hawthorne and Perry 2005) several reference architectural styles for self-healing systems presented and discussed that are described in Section 5.1.

4 Software Architecture

Structural design of the software system is a critical aspect of software design and it provides a high-level view of components and their communication. The definition of software architecture is provided in (Garlan and Shaw 1994) is "As the size and complexity of software systems increases, the design problem goes beyond the algorithms and data structures of the computation: designing and specifying the overall system structure emerges as a new kind of problem … This is the software architecture level of design." Another definition is given in (Bass et al. 2003) is "The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them." As shown in Fig. 1 software architecture establishes a relation between requirements and code (Bass et al. 2003).

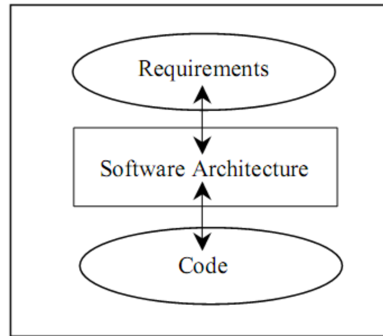

Fig. 1. Software Architecture as a relation (Bass et al. 2003)

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 dedicated to software quality attributes and the concept of coupling and cohesion respectively.

4.1 The Relationship between Software Architecture and Quality Attributes

Satisfying quality attributes in a system is largely determined by its architecture; so, understanding the relationship between the software architecture and quality attributes is important (Bass et al. 2003). Designing a system, regardless of how it supports of quality attributes may lead to redesign the system during its lifetime. Evaluating a system before its design is a good idea to avoid redesigning the system. The best software architecture of a system can be selected based on the amount of supporting of quality attributes.

Evaluating compositions of elements is necessary in evaluating architecture (Barbacci et al. 1995) and architectural styles are shown the building blocks of software architecture. Each software architecture style includes the factors that are interesting for each quality attribute; so evaluating a style means identifying the amount of supporting quality attributes. Architectural styles have a striking impression on system quality attributes such as reliability, maintainability and etc. So selecting the appropriate architectural style can be important in a system (Seo et al. 2008). In other words a quantitative impact of architectural style on quality attributes can help the software developers to select the best style.

4.2 Software Quality attributes

In this section, we explain the software quality attributes. Generally, there are different definitions for quality. In software engineering, software quality means there are no problems in the system, so the quality of a software product that has functional defects is low. There is a standard for the evaluation of software quality in (Iso 2001). In the first part of this standard the software quality attributes are classified as follows (Iso 2001): **Functionality**, **Reliability**, **Usability**, **Efficiency**, **Maintainability** and **Portability**. Self-healing systems due to the changes during the run time have other features.

Self-healing is the ability of discovering, detection and reaction against the failure. The main purpose of developing self-healing system is the enhancement in characteristics such as availability, durability, maintainability and reliability in the system (Ganek and Corbi 2003). In (Mazeiar Salehie and Tahvildari 2005) the attributes of self-adaptive systems and the relation of them with common quality attributes is discussed. Maintainability is one of key quality attributes in self-adaptive systems, and particularly in self-healing systems (Mazeiar Salehie and Tahvildari 2005).

Maintainability is defined in (Jay and Mayer 1990) as: "The ease with which a software system or component can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment." The ease of software correction is determined through: analyzability, changeability, stability and testability (Iso 2001). Software maintainability attribute depends on the modularity of software design, i.e. design with low coupling among modules and high cohesion of modules. As amount of coupling of the components decrease and their cohesion become more, the analyzability, changeability, stability, and testability of the software will be easier. The impact of cohesion and coupling on software maintainability is emphasized in (Pfleeger and Atlee 2006; Ohlsson and Alberg 1996). Cohesion and coupling are introduced in Section 4.3.

4.3 cohesion and coupling

Generally *coupling* is an attribute of a pair of modules that specifies the degree of interdependency between them (Yourdon and Constantine 1979). The coupling types are as follow (N. E. Fenton and Pfleeger 1998):

- *No coupling*: modules do not communicate with each other; that is they are totally independent of each other
- *Data coupling*: when modules share data with each other, by parameter passing
- *Stamp coupling:* when modules share a composite data structure by parameter passing
- *Control coupling*: when one module controls the process of another, by a control flag passing
- *Common coupling*: when some modules share the program global data
- *Content coupling:* when one module can change the internal workflow of another module. For example achieving local data of another module

In (N. Fenton and Melton 1990) consecutive numeric values from 0 to 5 are used for types of coupling and the experience from some software systems design was the basis of their work. These numeric values are given in Table 1.

Table 1. Numeric values for coupling (N. Fenton and Melton 1990)

Coupling type	**weight**	**symbol**
No coupling	0	w_0
Data coupling	1	w_1
Stamp coupling	2	w_2
Control coupling	3	w_3
Common coupling	4	w_4
Content coupling	5	w_5

As is defined in (Yourdon and Constantine 1979), cohesion means how a module needs to the other components to perform a job. The types of cohesion are as follows (N. E. Fenton and Pfleeger 1998):

- *Coincidental cohesion:* when a module performs the tasks that are not related to each other; the only connection between the parts is that they are gathered in a module.
- *Logical cohesion:* when a module perform the tasks that are logically related to each other; In other words, when parts of a module are collected to perform the same operation. (e.g. grouping all mouse and keyboard input handling routines).
- *Temporal cohesion*: when a module performs the tasks based on an event in a period. In other words, the parts of a module are placed in a group just because they perform tasks at a specified time.
- *Procedural cohesion*: sequence of activities that must be performed in a certain order. In other words, the parts of a module are placed in a group just because they run a certain sequence of operations.

- *Communicational cohesion*: when the parts of a module are grouped because they operate on the same data.
- *Sequential cohesion*: when the parts of a module are grouped because the output of one part, is the input to another.
- *Functional cohesion:* when all parts of a module, totally perform a single action. In other words, the parts of a module are placed in a group because all of them are working together on a single task.

In this article, consecutive numeric values from 0 to 6 were used for types of cohesion. These numeric values are given in Table 2.

Table 2. Numeric values for cohesion

Cohesion type	**weight**	**symbol**
Coincidental cohesion	0	c_0
Logical cohesion	1	c_1
Temporal cohesion	2	c_2
Procedural cohesion	3	c_3
Communicational cohesion	4	c_4
Sequential cohesion	5	c_5
Functional cohesion	6	c_6

5 Quantitative measurements of Architectural Styles for Self-healing Systems

As described in Section 4.1, architectural style has a striking impression on system quality attributes such as reliability, maintainability and etc. so selecting the appropriate architectural style can be important in a system (Seo et al. 2008). In other words, a quantitative impact of architectural style on quality attributes can help the software developers to select the best style.

As mentioned in Section 3, Self-healing is the ability of discovering, detection, and reaction against the failure. The main purpose of developing self-healing system is the enhancement in characteristics such as availability, durability, maintainability and reliability in the system (Ganek and Corbi 2003). In (Mazeiar Salehie and Tahvildari 2005) the attributes of self-adaptive systems and the relation of them with common quality attributes is discussed. Maintainability is one of key quality attributes in self-adaptive systems, and particularly in self-healing systems (Mazeiar Salehie and Tahvildari 2005). The impact of cohesion and coupling on software maintainability is emphasized in (Pfleeger and Atlee 2006; Ohlsson and Alberg 1996). With increasing coupling between components, the maintenance will reduce (Pfleeger and Atlee 2006), so the coupling and maintainability are inversely related. With increasing the cohesion between the components, the maintenance will increase, so the cohesion and maintainability are directly related together.

In this section, the measuring self-healing architectural styles from the cohesion and coupling viewpoint are given. These measurements are based on proposed metrics for coupling and cohesion. At first Hawthorne and Perry's architectural styles for self-healing systems are explained in Section 5.1. Then coupling and cohesion metrics are introduced in Section 5.2, and then the measurement of self-healing architectural styles is done based on these metrics. The measuring coupling and cohesion of self-healing architectural styles are given in Sections 5.3 and 5.4 respectively.

5.1 Architectural Styles for Self- healing Systems

Software architectural styles provide models for solving design problems in a way that every model describes components, responsibilities of each component and their relations (Shaw and Garlan 1996). A self-healing system must be able to modify its behavior according to changes in the environment; so such a system must be able to continuously monitor its environment and react accordingly. So self-healing system architectures should have three basic requirements (Hawthorne and Perry 2005): Detect the internal and external conditions that the system must respond to them by a *reflection mechanism*; Determine what actions should be taken in response to discovered conditions by a *reasoning mechanism*; perform the changes by a *configuration mechanism.* In (Hawthorne and Perry 2005) *runtime reflection components* are called *monitors*, and *configuration components* are called *configurators*. The model of self-healing architecture is shown in Fig. 2.

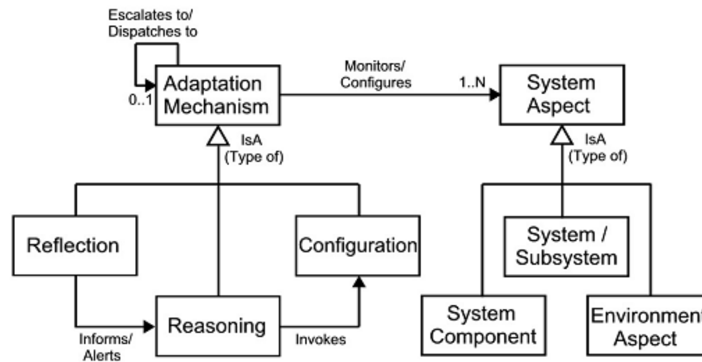

Fig. 2. Conceptual model of self-healing architecture (Hawthorne and Perry 2005)

In (Hawthorne and Perry 2005) several reference architectural styles for self-healing systems presented and discussed. Hawthorne and Perry's architectural styles for self-healing software systems are as follows:

- Aspect-peer-to-peer architectural style:

This style assigns a monitor component to each condition or aspect of the system that should be monitored; also each monitor has a corresponding configurator to perform the needed changes (Hawthorne and Perry 2005). The aspect peer-to-peer architectural style is shown in Fig. 3.

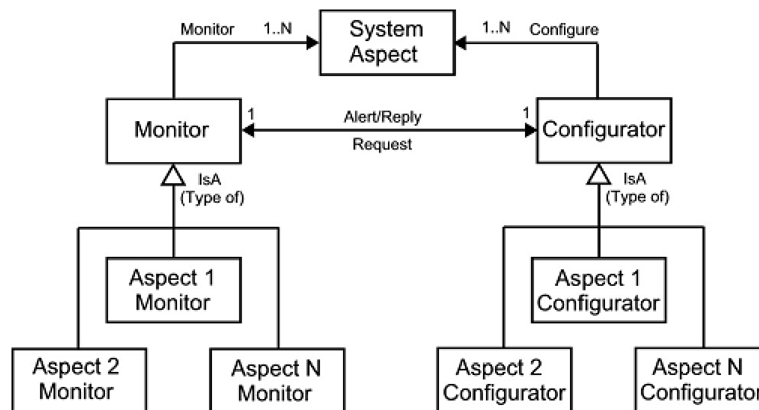

Fig. 3. Aspect Peer-to-Peer architectural style (Hawthorne and Perry 2005)

- Aggregator-escalator-peer architectural style:

In this style, monitors send their monitored data to high-level aggregator monitors; the high-level monitor packs the monitored data from the lower-level monitors into a composite package. Then the composite package is passed to a corresponding configurator (Hawthorne and Perry 2005). Fig. 4 shows the aggregator-escalator-peer architectural style.

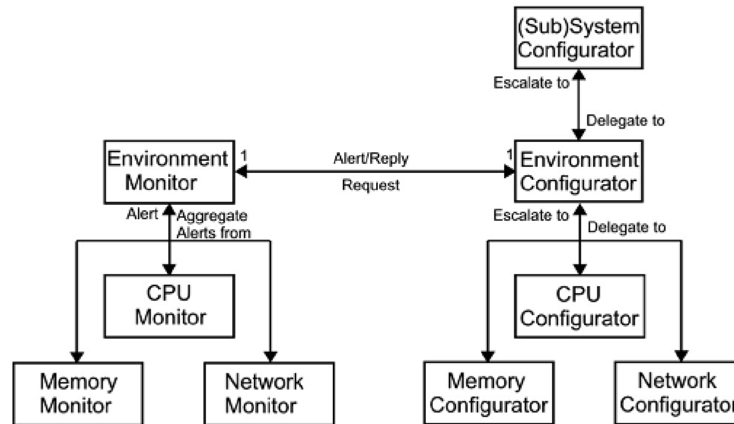

Fig. 4. Aggregator-Escalator-Peer architectural style (Hawthorne and Perry 2005)

- Chain-of-configurators architectural style:

In this style several configurator component are chained together. Configuration request is moved along the chain until it can be successfully processed (Hawthorne and Perry 2005). Fig. 5 shows the chain-of-configurators architectural style. This style consists of a single monitor to the entire system (Zhu et al. 2008).

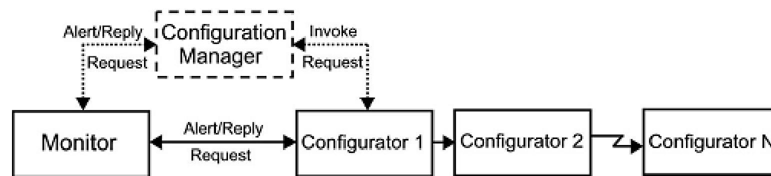

Fig. 5. Chain-of-configurators architectural style (Hawthorne and Perry 2005)

In this style, if one of configurators does not be able perform a request successfully the next configurator will try to do it. Another scenario of this style is that configurators perform tasks together and each configurator complete one part of the task (Zhu et al. 2008).

Hawthorne and Perry proposed architectural style for self-healing system but they didn't offer any formula for measuring the coupling and cohesion of these styles.

5.2 Coupling and Cohesion metrics

Although Hawthorne and Perry proposed architectural style for self-healing system but they didn't propose any formula for measuring the coupling and cohesion of these styles. In this article, the concept of "coupling among modules" is generalized to the coupling among software architecture components and is used to measure the amount of coupling of self-healing architectural styles. So the relation between monitors and configurators is considered as coupling. The coupling of a style is calculated due to the types of coupling among components and the number of involving components. As the strength of coupling among components become more, the understandability, correctness and maintainability of the components decrease (Pfleeger and Atlee 2006). We propose the Equation (1) to measure the coupling value of any self-healing architectural styles, where n is

the number of components that it can be number of monitors or configurators. SASCP is the amount of self-healing architectural styles coupling and CCP_i is the amount of coupling in i-th component. CCP_i is computed by Equation (2), where w_j is the weight of type j coupling according to Table 1 and k is the number of coupling in component i.

$$SASCP = \sum_{i=1}^{n} CCP_i \quad (1)$$

$$CCP_i = \sum_{j=1}^{k} (number\ of\ type\ j\ coupling \times w_j) \quad (2)$$

With increasing coupling between components, the maintenance will reduce (Pfleeger and Atlee 2006), so the coupling and maintainability are inversely related. Whatever the obtained value for the coupling calculated by (1) increases, the maintainability reduces.

In this article, the concept of "modules cohesion" is generalized to the cohesion of software architecture components and is used to measure the amount of cohesion of self-healing architectural styles. For example, the relation between parts of an element or between some elements that are grouped together to do a special task can be considered as cohesion. As the cohesion of components become more, the understandability, correctness, and maintainability of the components increase. We propose the Equation (3) to measure the cohesion of self-healing architectural styles, where n is the number of components. SASCH is the amount of self-healing architectural styles cohesion and CCH_i is the amount of cohesion in i^{th} component. CCH_i is computed by Equation (4), where c_j is the weight of type j cohesion according to Table 2 and k is the number of cohesion in component i.

$$SASCH = \sum_{i=1}^{n} CCH_i \quad (3)$$

$$CCH_i = \sum_{j=1}^{k} (number\ of\ type\ j\ cohesion \times c_j) \quad (4)$$

With increasing the cohesion between the components, the maintenance will increase, so the cohesion and maintainability are directly related together. Whatever the obtained value for the cohesion calculated by (3) increases, the maintainability increases.

The measuring coupling of self-healing architectural styles using equation (1) and (2) is described in Section 5.3. Then the measuring cohesion of self-healing architectural styles using equation (3) and (4) is described in Section 5.4.

5.3 Measuring the coupling of Self-healing Architectural Styles

a. Aspect-peer-to-peer architectural style:

This architectural style divides the system into a set of independent elements. A monitor and corresponding configurator belong to an element and there is no relation between elements. Thus, a strict peer-to-peer approach decreases coupling. It can be useful when configurators can decide based on corresponding monitor and do not need to information of other monitor's component. In this style, we assumed each configurator need to information from C number of monitors. Consequently, each configurator has Data Coupling with C number of monitor component. Additionally, monitors send a control flag to configurators and in this style we assumed, each monitor send a control flag to C number of configurators. Therefor, each monitor has Control Coupling with C number of configurator component. As a result, with respect to Equation (1) the coupling value of this style is equal to:

$$nCw_1 + nCw_3 \quad (5)$$

Where w_1 is the weight of data coupling and w_3 is the weight of control coupling. Also, n is the number of monitors or configurators.

b. Aggregator-escalator-peer architectural style:

This architectural style divides the system into a set of independent elements but in this case, the elements are not completely independent and are arranged in a hierarchy. Each monitor component sends information to the monitor in higher level, so there is Data Coupling. The higher-level monitors pack the information from the lower-level monitors and pass this composite data to a peer configurator; thus, there is Stamp Coupling. In addition, each higher-level monitor sends a control flag to its corresponding configurator; so, they have Control Coupling. Configurators at the low level use the information from high-level configurator, so there is Common Coupling between them. As a result, with respect to Equation (1) the coupling value in this style is equal to:

$$nw_1 + mw_2 + mw_3 + nw_4 \qquad (6)$$

Where, n is the number of low level monitors or the number of low level configurators and m is the number of subsystems that indicate the number of high level monitors. Also, w_1 is the weight of data coupling, w_2 is the weight of stamp coupling, w_3 is the weight of control coupling and w_4 is the weight of common coupling,

c. Chain-of-configurators architectural style:

In this style, configurators perform tasks together and each configurator complete one part of the task; then passes the data to the next configurator until all the actions have been done successfully. (Zhu et al. 2008). Therefore, each configurator has a Data Coupling with its previous configurator but the last one does not send any information. As a result, with respect to Equation (1) the coupling value in this style is equal to:

$$(n-1)w_1 \qquad (7)$$

Where w_1 is the weight of data coupling and n is the number of configurators. In this style there is loose coupling between components.

We measured the coupling of Self-healing Architectural Styles and the results are shown in Table 3. The third column shows the coupling value by replacing the weights based on Table 1.

Table 3. Coupling formulas of Self-healing Architectural Styles

Architectural style	Coupling formula	Coupling value
Aspect-peer-to-peer	$nCw_1 + nCw_3$	$4Cn$
Aggregator-escalator-peer	$nw_1 + mw_2 + mw_3 + nw_4$	$5n + 5m$
Chain-of-configurators	$(n-1)w_1$	$n-1$

5.4 Measuring the cohesion of Self-healing Architectural Styles

a. Aspect-peer-to-peer architectural style:

In this style, each element includes a monitor and a corresponding configurator and there is no type of cohesion in it.

b. Aggregator-escalator-peer architectural style:

In this style for each subsystem of the system, there is higher-level monitor and lower level monitors that are related to a special aspect of the system. Each of lower level monitors, supervise one part of the subsystem and all the monitors are logically related to it; so there is Logically Cohesion between

monitors. All the lower level configurators use information from high-level configurators; so there is Communicational Cohesion between them. As a result, with respect to Equation (3) the cohesion value in this style is equal to:

$$nc_1 + nc_4 \quad (8)$$

Where c_2 is the weight of logical cohesion and c_5 is the weight of communicational cohesion and n is the number of monitors or configurators.

c. Chain-of-configurators architectural style:

In this style, configurators are chained together and control passes from one configurator to the next; so there is Procedural Cohesion between them. Additionally in this style, configurators are involved in activities such that output data from one configurator serves as input data to the next; so there is Sequential Cohesion between them. As a result, with respect to Equation (3) the cohesion value in this style is equal to:

$$(n-1)c_3 + (n-1)c_5 \quad (9)$$

Where c_4 is the weight of procedural cohesion and c_6 is the weight of sequential cohesion and n is the number of configurators.

We measured the cohesion of Self-healing Architectural Styles and the results are shown in Table 4. The third column shows the cohesion value by replacing the weights based on Table 2.

Table 4. Cohesion formulas of Self-healing Architectural Styles

Architectural style	**Cohesion formula**	**Cohesion value**
Aspect-peer-to-peer	Zero	Zero
Aggregator-escalator-peer	$nc_1 + nc_4$	$5n$
Chain-of-configurators	$(n-1)c_3 + (n-1)c_5$	$8(n-1)$

6 Case study: Wireless Body Area Network[1]

Nowadays, in many countries, costs of health care are rising sharply and this leads to introduction of new systems based on new technology to protect the body. New developments lead to construction the Wireless Body Area Networks system or WBAN. In these networks, various sensors are attached on clothing or body or even implanted under the skin are. These sensors should be able to send medical information to an external server (outside the body of patient). In the server data can be stored, maintained and analyzed (Latré et al. 2011). Use a wired connection to the development of these systems makes it difficult and expensive. However, developing these systems using wireless connections will be more simple, cheaper and easier to maintain (Cypher et al. 2006). In addition, using this system, patients do not require a lot of time to stay in the hospital.

A WBAN consists of small and smart devices that is connected to the body or implanted in the body and is able to connect wirelessly. The devices continuously monitor the body and give feedback to the user or medical personnel (Latré et al. 2011). In addition, measurement in a long period of time can enhance the quality of analyzing recorded data (Park and Jayaraman 2003).

[1] WBAN

Generally, there are two types of devices in a WBAN system: **Sensors** and **Actuators**. Sensors are used to measure specific parameters of the human body and can be external or internal. Examples of these sensors are as follows: sensors to measuring the heartbeat, body temperature measurement or sensors that record the ECG for a long time. Actuators perform specific actions with respect to the information received from the sensors or through user interaction. For example, an actuator equipped with a built-in reservoir and pump, so it can administer the correct dose of insulin that should be given to diabetics based on the glucose level measurements. Additionally, a personal device such as a phone can be used to interaction with the user (Latré et al. 2011). Now, the precise definition of the various components of the wireless network is discussed. These components are (Latré et al. 2011):

(Wireless) Sensor node: a device that responds to physical stimulation, gather information with respect to the stimulation. If necessary, the data can be processed and will be sent to an external device wirelessly. This element includes various components like hardware sensor, a power supply unit, a processor unit, memory and a transmitter/receiver.

(Wireless) Actuator node: a device that act actions to improve the health care of patient. This element act based on information gathered by sensors. The components of an actuator and sensor are similar. The components of an actuator include actuator hardware, a power supply unit, processor, memory and a transmitter/receiver.

(Wireless) Personal Device (PD): A device that collects data from all the sensors and actuators and informs the user via an external element.

The several cases where the patient should be placed under the control are introduced in Section 6.1. Measuring the coupling and cohesion of WBANs system are described in Section 6.2.

6.1 Patient Monitoring

More than 30% of the main cause of death is cardiovascular disease and most of these deaths can be prevented with health care (Kephart and Walsh 2004). A WBAN allows continuous monitoring of physiological parameters of the body. With this system, the patient can be in the hospital, at home or on the move and does not need to stay in bed and be able to move. An example of a WBAN system for monitoring the patient's body is shown in Fig. 6. As can be seen in Fig. 6, there are many sensors in clothing, on the body or has been placed under the skin for a person and measure temperature, blood pressure, heart rate, ECG, EEG, respiration rate. The sensors are equipped with actuators, which perform drug delivering to patient body. Injection time can be pre-determined, triggered by an external source or by monitoring the parameters. An example is the monitoring of blood glucose levels in diabetic patients; if, the sensor shows a sudden drop in glucose level, a signal will be sent to the corresponding actuator to begin insulin injections. As a result, the patient will experience less pain (Krames 2002).

In addition, a WBAN system can be used to help the disabled persons. For example, the paralyzed person can be equipped with sensors to help him. These sensors can determine person's location. Also actuators can help him to move (Li and Kohno 2008).

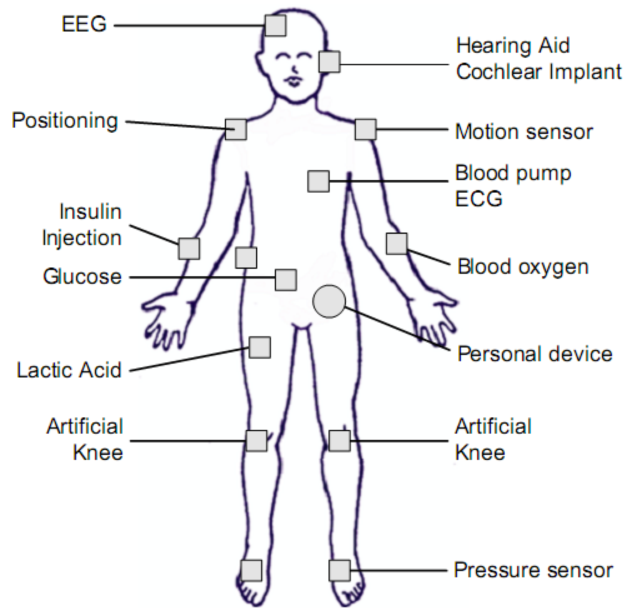

Fig. 6. Patient monitoring in a wireless body area network (Li and Kohno 2008)

6.2 Measuring the coupling and cohesion of Wireless Body Area Networks system

In this section, the design of a WBAN system with self-healing architectural styles is introduced. Then using the formula described in Section 5.3 and 5.4, we measured the coupling and cohesion of the WBANs system with different styles.

As we introduced in Section 5.1, implementing the system with self-healing architectural styles needs to have two components: **monitor** and **configurator**. In WBANs system, *sensors* act as *monitors* and *actuators* can act as *configurators*.

In this paper, we have considered a WBAN system whose sensors are: EEG sensor, auditory sensor, motion sensor, position sensor, ECG sensor, heart rate sensor, blood sensor, fingertip sensor, the sensor detects blood pressure, the sensor detects blood oxygen. Also the actuators of this system includes hearing aid cochlear implant, artificial knee, artificial arm, blood pump, insulin injection, a device for injection to increase or decrease blood pressure, blood oxygen, and pacemaker.

The design of this system by aspect-peer-to-peer architectural style is shown in Fig. 7. Audiometry sensor sends data to the hearing aid cochlear implant actuator and this actuator requires no additional data from other sensors; so, the audiometry sensor and its corresponding actuator make an element. The artificial knee receives data from EEG sensor; so artificial knee and EEG sensor make an element. Moreover, the artificial arm receives data from EEG sensor; so, the artificial arm and EEG sensor make an element. Additionally, the artificial knee and the artificial arm need to receive data from motion and positioning sensors. The blood pump is related to ECG sensor and both of them make an element. The blood pump needs to receive data from heartbeat sensor. The need for insulin is recognized by fingertip sensor; so, the insulin injection actuator and fingertip sensor make an element. The insulin injection actuator needs to receive data from motion, position, and blood sensors. The actuator that injects a liquid material to increase (or decrease) the blood pressure is related to blood pressure detection sensor and both of them make an element. Furthermore, this actuator needs to receive data from motion and position sensors. The blood oxygen

actuator acts based on data received from the blood oxygen detector sensor and both of them make an element. Pacemaker receives data from the heartbeat sensor and both of them make an element.

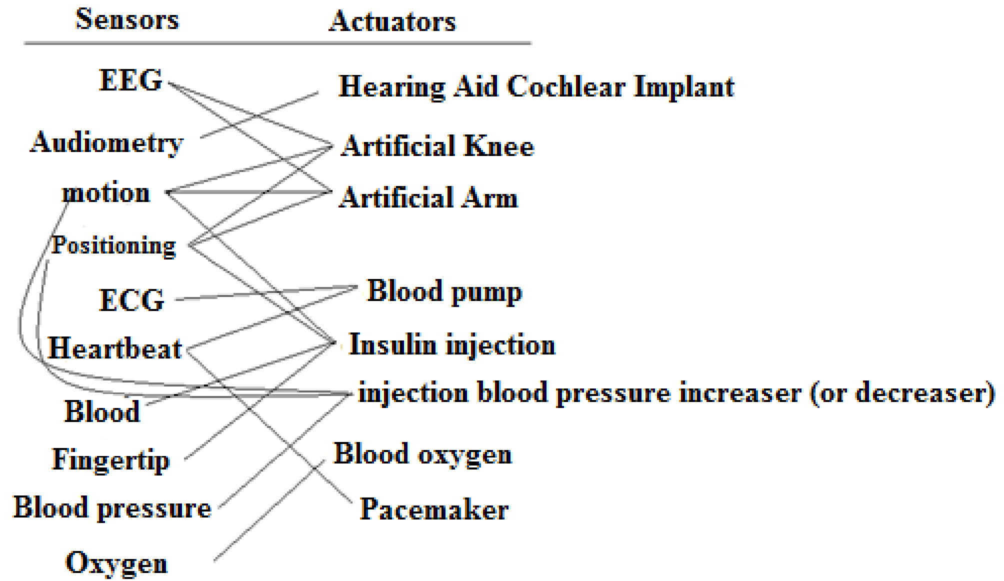

Fig. 7. Design of a WBAN system by Aspect-peer-to-peer architectural style

Now we deal with measuring the coupling of aspect-peer-to-peer architectural style in a WBAN using Relation (5). The artificial knee actuator needs data received from its respective sensor and 2 other; so, it has two data coupling. The blood pump actuator acts based on data received from its respective sensor and another one; so, it has one data coupling. We computed data coupling for other actuators in the same way. The EEG sensor sends a control flag to its respective actuator and another one; so, it has one control coupling. We computed control coupling for other actuators in the same way. Finally, the coupling of aspect-peer-to-peer architectural style is:

$$SASCP = w_1(2 + 2 + 1 + 3 + 2) + w_3(1 + 3 + 3 + 1) = 1 \times 10 + 3 \times 8 = 34$$

As described in Section 5.4, there is no type of cohesion in aspect-peer-to-peer architectural style and the amount of cohesion in this style is zero.

To design the WBAN with aggregator-escalator-peer architectural style, first we identify interdependent parts of the system. The design of WBAN by aggregator-escalator-peer architectural style is shown in Fig. 8.

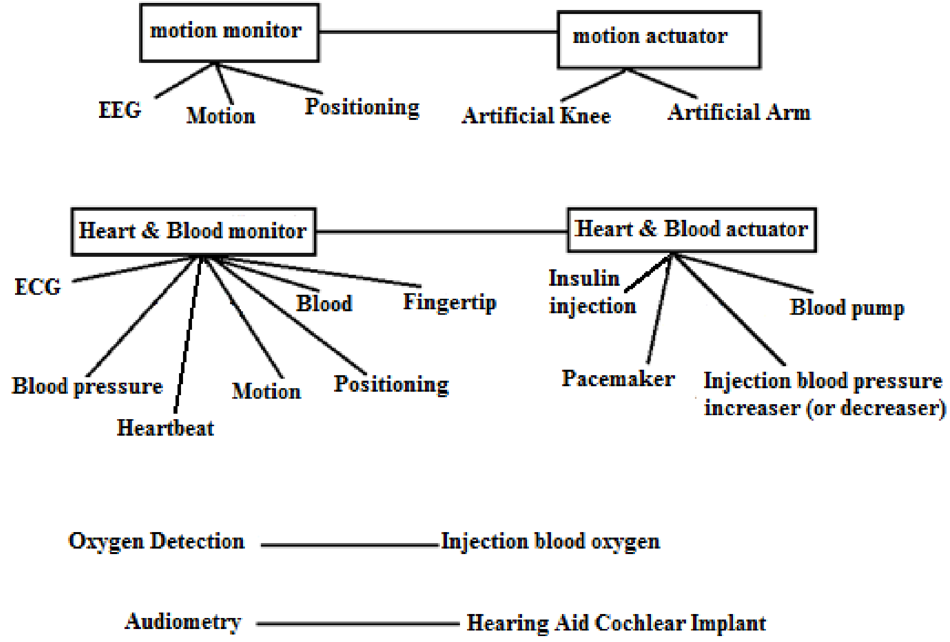

Fig. 8. The design WBAN system by Aggregator-escalator-peer architectural style

Now, we measure the coupling of aggregator-escalator-peer architectural style in WBAN using the (6). Each monitor component sends its data to the higher-level monitor; therefore, there is a data coupling between them. As a result, the amount of data coupling will be equal to the number of low-level monitors. Furthermore, higher-level monitors collect all received data from low-level monitors and send this composite data to the corresponding actuator; so, they have the stamp coupling. Moreover, each high-level monitor sends a control flag to the corresponding actuator to report a new event; therefore, they have the control coupling. As a result, amount of the stamp coupling and the control coupling will be equal to the number of high-level monitors. Low-level actuators use high-level actuator's data; so, they have the common coupling. Finally, the coupling of aggregator-escalator-peer architectural style is:

$$SASCP = 10w_1 + 2(w_2 + w_3) + 6w_4 = 10 + 10 + 24 = 44$$

The cohesion of aggregator-escalator-peer architectural style in the WBAN is computed using Relation (8). Each low-level monitor controls a particular aspect of system; so, the low-level monitors logically are related to high-level monitors and there is a logical cohesion between them. Therefore, amount of the logical cohesion is equal to the number of low-level monitors. On the other, all low-level actuators use high-level actuator's data; so, there is a communicational cohesion between them. Therefore, amount of the communicational cohesion is equal to the number of low-level actuator. Finally, the cohesion of the aggregator-escalator-peer architectural style is:

$$SASCH = c_1(3 + 7) + c_4(2 + 4) = 10 + 4 \times 6 = 34$$

The design of the WBAN by thechain-of-configurators architectural style is shown in Fig. 9.

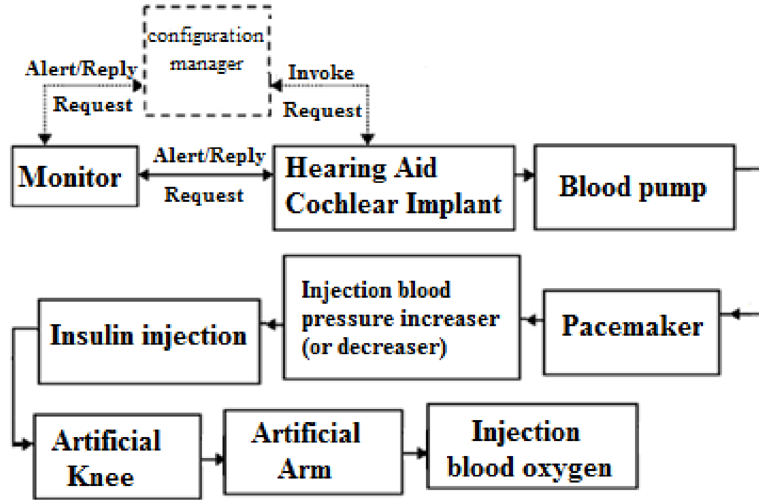

Fig. 9. Design of the WBAN system by Chain-of-configurators architectural style

Now we measure amount of coupling of the chain-of-configurators architectural style in the WBAN using Relation (7). In this style, multiple actuators are chained together in form of a linear structure and each actuator sends the data to the next actuator; so, each actuator has one data coupling with the previous actuator. Finally, amount of coupling of chain-of-configurators architectural style is:

$$SASCP = (n-1)w_1 = (8-1)w_1 = 7 \times 1 = 7$$

Amount of cohesion of the chain-of-configurators architectural style for the WBAN is computed using Relation (9). In this style, actuators perform a specific flow of operations; so, there is a procedural cohesion between them. In addition, the output of an actuator is used as input to the next actuator; so, they have a sequential cohesion. Finally, amount of cohesion of the chain-of-configurators architectural style is:

$$SASCH = (8-1)c_3 + (8-1)c_5 = 7 \times 3 + 7 \times 5 = 21 + 35 = 56$$

Amount of coupling and cohesion for different styles in the WBAN system is shown in Fig. 10. As shown in Fig. 10, the chain-of-configurators has the lowest coupling amount and the most cohesion amount in comparison with other styles; so. It is the best style from *maintainability* viewpoint.

7 Conclusions

A self-healing software system can adapt to the changes occurred in its environment at run-time. The software architecture of a system is the set of structures that are necessary to justify the software system. Generally, according to the software engineering field, the software quality discusses problems of a system; so, a software product of good quality means that its functional defects are low.

In this article, we measured amount of coupling and cohesion for the self-healing software architectural style. The *maintainability* of a system is closely related to coupling and cohesion principles. We proposed metrics for measuring coupling and cohesion of the system in order to quantify self-healing architectural styles. To show effectiveness of our metrics, we applied them to the Wireless Body Area Networks as a case study. We compared mount of coupling between three

self-healing software architectural styles (Fig. 10) based on the formulae in Table 3 and concluded that the chain-of-configurators architectural style has the lowest coupling amount; so, it is the best style from the coupling viewpoint. Moreover, by comparing amount of cohesion of three self-healing software architectural styles based on the formulae in Table 4 (Fig. 10), we find that the chain-of-configurators architectural style has the most cohesion amount; so, it is the best style from the coupling viewpoint. Therefore, the chain-of-configurators architectural style is the best one from the *maintainability* view.

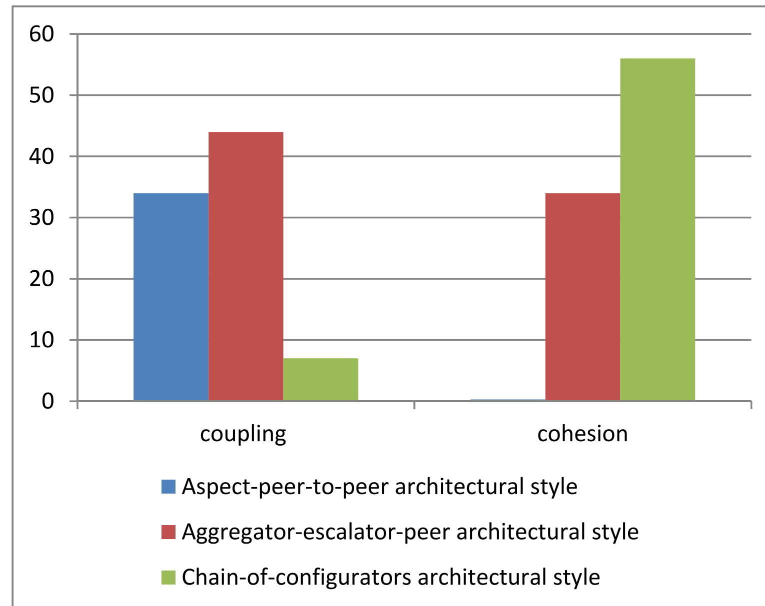

Fig. 10. Amount of coupling and cohesion for different styles in the WBAN system

References

Babaoglu, O., Jelasity, M., Montresor, A., Fetzer, C., Leonardi, S., van Moorsel, A., et al. (2005). *Self-star properties in complex information systems: conceptual and practical foundations* (Vol. 3460): Springer.

Barbacci, M., Klein, M. H., Longstaff, T. A., & Weinstock, C. B. (1995). Quality Attributes. DTIC Document.

Bass, L., Clements, P., & Kazman, R. (2003). *Software architecture in practice*: Addison-Wesley Professional.

Cheng, B., de Lemos, R., Giese, H., Inverardi, P., Magee, J., Andersson, J., et al. (2009). Software engineering for self-adaptive systems: A research roadmap. *Software Engineering for Self-Adaptive Systems*, 1-26.

Cheng, S. W., & Garlan, D. Handling uncertainty in autonomic systems. In *Proceedings of the International Workshop on Living with Uncertainties (IWLU'07), 2007*

Cypher, D., Chevrollier, N., Montavont, N., & Golmie, N. (2006). Prevailing over wires in healthcare environments: benefits and challenges. *Communications Magazine, IEEE, 44*(4), 56-63.

Dashofy, E. M., Van der Hoek, A., & Taylor, R. N. Towards architecture-based self-healing systems. In *Proceedings of the first workshop on Self-healing systems, 2002* (pp. 21-26): ACM

Fenton, N., & Melton, A. (1990). Deriving structurally based software measures. *Journal of Systems and Software, 12*(3), 177-187.

Fenton, N. E., & Pfleeger, S. L. (1998). *Software metrics: a rigorous and practical approach*: PWS Publishing Co.

Ganek, A. G., & Corbi, T. A. (2003). The dawning of the autonomic computing era. *IBM systems Journal, 42*(1), 5-18.

Garlan, D., & Shaw, M. (1994). An introduction to software architecture.

Hawthorne, M. J., & Perry, D. E. Architectural styles for adaptable self-healing dependable systems. In *Proceedings of IEEE/ACM International Conference on Software Engineering (ICSE 2005), 2005*

Iso, I. (2001). IEC 9126-1: Software Engineering-Product Quality-Part 1: Quality Model. *Geneva, Switzerland: International Organization for Standardization*.

Jay, F., & Mayer, R. (1990). Ieee standard glossary of software engineering terminology. Office.

Kephart, J. O., & Chess, D. M. (2003). The vision of autonomic computing. *Computer, 36*(1), 41-50.

Kephart, J. O., & Walsh, W. E. An artificial intelligence perspective on autonomic computing policies. In *Policies for Distributed Systems and Networks, 2004. POLICY 2004. Proceedings. Fifth IEEE International Workshop on, 2004* (pp. 3-12): IEEE

Krames, E. (2002). Implantable devices for pain control: spinal cord stimulation and intrathecal therapies. *Best Practice & Research Clinical Anaesthesiology, 16*(4), 619-649.

Laddaga, R. (2001). Active software. *Self-Adaptive Software*, 11-26.

Laddaga, R., Robertson, P., & Shrobe, H. E. (1997). Self-adaptive software. *Proposer Information Pamphlet BAA*(98-12).

Latré, B., Braem, B., Moerman, I., Blondia, C., & Demeester, P. (2011). A survey on wireless body area networks. *Wireless Networks, 17*(1), 1-18.

Li, H. B., & Kohno, R. (2008). Body area network and its standardization at IEEE 802.15. BAN. *Advances in Mobile and Wireless Communications*, 223-238.

Magee, J., Dulay, N., Eisenbach, S., & Kramer, J. (1995). Specifying distributed software architectures. *Software Engineering—ESEC'95*, 137-153.

Neti, S., & Muller, H. A. Quality criteria and an analysis framework for self-healing systems. In *Software Engineering for Adaptive and Self-Managing Systems, 2007. ICSE Workshops SEAMS'07. International Workshop on, 2007* (pp. 6-6): IEEE

Ohlsson, N., & Alberg, H. (1996). Predicting fault-prone software modules in telephone switches. *Software Engineering, IEEE Transactions on, 22*(12), 886-894.

Oreizy, P., Gorlick, M. M., Taylor, R. N., Heimhigner, D., Johnson, G., Medvidovic, N., et al. (1999). An architecture-based approach to self-adaptive software. *Intelligent Systems and Their Applications, IEEE, 14*(3), 54-62.

Parashar, M., & Hariri, S. (2005). Autonomic computing: An overview. *Unconventional Programming Paradigms*, 97-97.

Park, S., & Jayaraman, S. (2003). Enhancing the quality of life through wearable technology. *Engineering in Medicine and Biology Magazine, IEEE, 22*(3), 41-48.

Pfleeger, S. L., & Atlee, J. M. (2006). *Software Engineering: Theory and Practice*: Prentice Hall.

Salehie, M., & Tahvildari, L. Autonomic computing: emerging trends and open problems. In *ACM SIGSOFT Software Engineering Notes, 2005* (Vol. 30, pp. 1-7): ACM

Salehie, M., & Tahvildari, L. (2009). Self-adaptive software: Landscape and research challenges. *ACM Transactions on Autonomous and Adaptive Systems (TAAS), 4*(2), 14.

Seo, C., Edwards, G., Malek, S., & Medvidovic, N. A framework for estimating the impact of a distributed software system's architectural style on its energy consumption. In *Software Architecture, 2008. WICSA 2008. Seventh Working IEEE/IFIP Conference on, 2008* (pp. 277-280): IEEE

Sharafi, S. M., Ghazvini, G. A., & Emadi, S. An analytical model for performance evaluation of software architectural styles. In *Software Technology and Engineering (ICSTE), 2010 2nd International Conference on, 2010* (Vol. 1, pp. V1-394-V391-398): IEEE

Shaw, M., & Garlan, D. (1996). Software architecture: perspectives on an emerging discipline.

Wang, W.-L., Pan, D., & Chen, M.-H. (2006). Architecture-based software reliability modeling. *Journal of Systems and Software, 79*(1), 132-146.

Wang, W.-L., Wu, Y., & Chen, M.-H. An architecture-based software reliability model. In *Dependable Computing, 1999. Proceedings. 1999 Pacific Rim International Symposium on, 1999* (pp. 143-150): IEEE

Yourdon, E., & Constantine, L. L. (1979). *Structured design: fundamentals of a discipline of computer program and systems design*: Prentice-Hall, Inc.

Zhu, Q., Lin, L., Kienle, H. M., & Muller, H. Characterizing maintainability concerns in autonomic element design. In *Software Maintenance, 2008. ICSM 2008. IEEE International Conference on, 2008* (pp. 197-206): IEEE