EVALUATION OF THE SOFTWARE ARCHITECTURE STYLES FROM MAINTAINABILITY VIEWPOINT

Gholamreza ShahMohammadi[1]

[1]Department of Information Thechnology,
Olum Entezami University-Amin, Tehran, Iran
shah_mohammadi@yahoo.co.uk

ABSTRACT

In the process of software architecture design, different decisions are made that have system-wide impact. An important decision of design stage is the selection of a suitable software architecture style. Lack of investigation on the quantitative impact of architecture styles on software quality attributes is the main problem in using such styles. So, the use of architecture styles in designing is based on the intuition of software developers. The aim of this research is to quantify the impacts of architecture styles on software maintainability. In this study, architecture styles are evaluated based on coupling, complexity and cohesion metrics and ranked by analytic hierarchy process from maintainability viewpoint. The main contribution of this paper is quantification and ranking of software architecture styles from the perspective of maintainability quality attribute at stage of architectural style selection.

KEYWORDS

Maintainability Evaluation, Software Architecture, Architecture Style, Coupling, Complexity, Cohesion

1. INTRODUCTION

Functionality may be achieved using a number of possible structures [1], so software architecture styles (SASs) are selected based on the amount of their support from quality attributes. SASs present models to solve the problem of designing the software architecture in a way that each model describes its components, responsibilities of the components and the way they cooperate [2]. Architectural decisions made early in the design process are a critical factor in the successful development of the system. In particular, the selection of an appropriate architectural style has a significant impact on various system quality attributes [3]. Since quantitative impacts of SASs on quality attributes have not been studied so far [4], their applications are not systematic [5]. In other words, the current use of SASs in design is ad-hoc and based on the intuition of software developers.

A method has been shown in [6], to map an architectural style into a relational model that can be checked for various style properties such as consistency of style. In [7], two graph-based approaches have been shown and compared to the specification and modeling of dynamic software architectures. The impact of a distributed software system's architectural style on the system's energy consumption has been estimated in [3]. A method for specifying the relation between six SASs and quality attributes such as maintainability has been proposed in [8]. The relationship between the quality attributes, design principles and some SASs has been specified in [8] using a tree-based framework. In [4], the impacts of SASs on quality attributes are determined based on the description of style in [2]. The methods offered in [4] and [8] are not able to determine the amount of style support from quality attributes, do not offer quantitative results about their maintainability, and are not precise. SASs are evaluated in [9] from maintainability viewpoint based on the scenario-based evaluation method that is less precise, less reliable and less analyzable as compared to the measurement-based evaluation method utilized in this paper.

In [10], the performance of three SASs has been investigated through simulation-based evaluation method. Implicit/invocation style has been verified in [11], by model checking method.

In this study, the quantitative impact of SASs on software maintainability, one of the important quality attributes required by all software, is determined based on the measurement-based evaluation of SASs. The SASs evaluated include Repository (PRS), Blackboard (BKB), Pipe and Filter (P/F), Layered (LYD), Implicit/Invocation (I/I), Client/Server(C/S), Broker (BRK) and Object-Oriented (OO), which have been introduced in [2], [12].

Software architecture evaluation methods include: 1) scenario-based evaluation, 2) simulation-based evaluation, 3) measurement-based evaluation and 4) mathematic model-based evaluation.
Measurement-based evaluation method uses metrics to measure software architecture. Metrics evaluate internal attributes of software (e.g. coupling). External attributes (e.g. maintainability) reflect those properties that are desirable for the software user and usually are evaluated by internal attributes. It is believed that there is a relationship between internal and external quality attributes. This relationship is based on theoretical models and empirical study [13], [14]. There is a general agreement in software community that modularity has an influence on external attributes such as maintainability [15]. Therefore, in this paper, we use coupling, complexity and cohesion metrics to quantify the impact of SASs on software maintainability. These metrics are essential in evaluation of software design quality and their effects on maintainability have been extensively investigated [15]-[18].

The advantage of measurement-based evaluation as compared to scenario-based evaluation is that the evaluation would be easier and more precise, if there are appropriate metrics. In addition, it does not have the problems of scenario-based evaluation, namely the dependency of the results on the scenarios used, and extensive participation of the expert. As a result, the problem is evaluated more comprehensively.

Multi-criteria decision-making methods are used in the ranking problem of SASs. These methods are in three categories: 1) scoring, 2) compromise and 3) concordance [19]. Analytic hierarchy process (AHP) [20] is one of the most comprehensive multi-criteria decision making methods. It structures the problem as a hierarchy and provides a means of decomposing the problem into a hierarchy of sub problems that can more easily be comprehended and subjectively evaluated. AHP reflects the way people think and behave. It also considers different quantitative and qualitative criteria in the problem and provides sensitivity analysis on the criteria and sub-criteria. The AHP has been proven a theoretically sound, market-tested and accepted methodology.
In this paper, to rank SASs based on the results of measurement-based evaluation of SASs, AHP method is used.

This paper is structured as follows: Section 2 discusses software maintainability and its measurement metrics. Section 3 explains the quantitative measurement of SASs. Section 4 deals with the ranking of SASs. Finally, Section 5 presents the conclusion.

2. SOFTWARE MAINTAINABILITY AND ITS MEASUREMENT METRICS

The main objective of any software is to offer desired services according to the predetermined quality level. There is a strong connection between many quality attributes and the software architecture of the software system. The architecture defines the overall potential that a software system possesses to fulfil certain quality attributes. Software are often redesigned not for the deficiency in the functionality, but due to difficulty in maintenance, port or scale [21].

Maintainability is the capability of the software product to be modified [22]. Modifications may include corrections, improvements or adaptations of the software to changes in the environment and in the requirements and functional specifications. The ease of software correction is determined through: 1) analyzability, 2) changeability, (3) stability and (4) testability [22].

A close look at software maintainability attributes reveals that provision of each characteristic depends on the amount of modularity of software design, design with low coupling among modules, low complexity of the modules and high cohesion of modules. Therefore, the less is the amount of coupling and complexity of the components and the more their cohesion, the easier will be the analyzability, changeability, stability and testability of the software. Various researches emphasize the impact of complexity, cohesion of components and coupling among components metrics on software maintainability [15]-[18].

2.1. Coupling Metric

High interaction of modules makes the understanding and modification of the modules more difficult [15]. The more independent the components, the easier their understanding, modification and maintainability [16]. Coupling is a complex concept that has been categorized by Yourdon and Constantine [23] as: 1) Data coupling, 2) Stamp coupling, 3) Control coupling, 4) Shared coupling and 5) Content coupling.

In this work, we generalize the "coupling among modules" concept to the coupling among software architecture components and use it to measure the amount of coupling of SASs. Components of SASs investigated in this work have three coupling types: data, stamp and shared quantified based on Table 1. In [24] also consecutive numeric values from 1 to 5 were used and the basis of such assignment was the experience from some software systems

Table 1.Type of Components Coupling

Row	Coupling type	Symbol	Weight
1	Data	w_1	1
2	Stamp	w_2	2
3	Common	w_3	4

designs. Regarding the coupling metric, SASs are investigated in terms of the type of coupling among the components and the number of components involved in the coupling. The more the strength of coupling among components and the more the number of components involved in the coupling, the less the understandability, correction and maintainability of the components [15]. To measure the coupling value of any SAS, (1) is used that is Euclidean norm, where n is the number of style components, SCP is the amount of SAS coupling and CCP_i is the amount of coupling of the i-th component. CCP_i is computed by (2), where NCT_j is the number of type j couplings, w_j is the weight of the corresponding coupling type and p is the number of coupling of the component i:

$$SCP = \sqrt{\sum_{i=1}^{n} CCP_i^2} \quad (1)$$

$$CCP_i = \sum_{j=1}^{p} NCT_j . w_j \quad (2)$$

2.2. Complexity Metric

Complexity value of SASs is computed by (3) where, SCM is the complexity of SAS, n is the number of style components and CCM_i is the amount of complexity of the i-th component. CCM_i is computed by (4), using the module evaluation metric of Shepperd et al [25], where $f_{in}(i)$ is the fan-in of component i and $f_{out}(i)$ is the fan-out of component i.

$$SCM = \sqrt{\sum_{i=1}^{n} CCM_i^2} \quad (3)$$

$$CCM_i = [f_{in}(i)*f_{out}(i)]^2 \quad (4)$$

$f_{in}(i)$ and $f_{out}(i)$ are computed by (5) and (6). In (5), Nc_i is the sum of the number of invocations of component i by other components and Nr_i is the number of data that component i has retrieved from the repository. In (6), Nce_i is the number of other components called by component i and Nu_i is the number of the repository data updated by component i. A component that controls a lot of components usually performs various functions and so it will have a high complexity [15],[26].

$$f_{in}(i)=Nc_i+Nr_i \quad (5)$$

$$f_{out}(i)=Nce_i+Nu_i \quad (6)$$

2.3. Cohesion Metric

The cohesion of a module is the extent to which its individual components are needed to perform the same task. Types of cohesion are: 1) Coincidental, 2) Logical, 3) Temporal, 4) Procedural, 5) Communicational, 6) Sequential and 7) Functional [23]. The cohesion type of every component is computed based on the available information of functionality of each component of SASs regarding the definition of the type of component cohesion in Table 2 [26].

In this work, we generalize the "modules cohesion" concept to the cohesion of software architecture components and use it to measure the amount of cohesion of SASs. Our investigations showed that cohesion of SASs in component are of three types: Functional, Communicational and Logical, which are quantified based on Table 2.

Table 2. Type of Components Cohesion [26]

Cohesion type	**Description**	**Symbol**	**Weight**
Logical	Component performs multiple functions, and in each calling, one of them is executed	C_1	1
Communicational	Component refers to the same data set and/or creates the same data set	C_2	2
Functional	Component performs a single well-defined function	C_3	3

Since every component i may have different type of cohesion (C_j), so the cohesion type of component i, CCHi, is computed by (7). Finally, cohesion of SASs is computed by (8).

$$CCH_i = \arg\min_j C_j \quad (7)$$

$$SCH = \sqrt{\sum_{i=1}^{n} CCH_i^2} \quad (8)$$

3. Quantitative Measurement of SASs

In this section, SASs are measured from the viewpoint of maintainability based on coupling, complexity and cohesion metrics.

The effect of software size on SASs ranking is taken into account in the computations of this section. In object-oriented style, the number of objects (no) and in other SASs, the number of components (n) correspond the software size. So in the evaluations done in this section, the number of SASs components is considered as 3, 4, 5, 6, 7, 8 and 9 and the number of classes in object-oriented style is considered accordingly as 21, 28, 35, 42, 49, 56 and 63.

3.1. Measuring the Coupling of SASs

In this section, the coupling formula of every SAS is computed using (1) to (2).

A. Repository style. In this style, all components have common coupling with the repository. Therefore, any change in the repository affects them. If the number of components in the repository style is n, then the number of couplings in this style will be n as well. Thus coupling of repository style is obtained from $\sqrt{n}.w_3$.

B. Blackboard style. The control component has a common coupling with the blackboard and has data coupling with the knowledge resources. Therefore, the control component has one common coupling and n data coupling while the knowledge resources have a common coupling with the blackboard. Thus, the coupling of the control component is n.w1+ w3 and the coupling of each knowledge resource is w3. Then coupling of this style is obtained from $\sqrt{(n.w_1+w_3)^2+n.w_3^2}$.

C. Pipe and filter style Every filter (component) has a stamp coupling with the next filter while the last filter has no coupling with any other filter. The number of couplings is n-1, and regarding the coupling type, the coupling of this style is obtained from $\sqrt{n-1}\,w_2$

D. Layered style The coupling type of every layer (component) with its lower layer is data. Considering the fact that coupling is two way, the last and first layers have only one coupling while other layers have two couplings. So for n layer, the coupling of this style is obtained from $\sqrt{4(n-2).W_1^2+2.W_1^2}$.

E. Implicit invocation style. If, in average, n/2 of components publish the events that are favored by n/2 of the components, the coupling type of the event publisher component with the dispatcher component is data. If an event occurs, the dispatcher component invokes the interested components, so the coupling type of the dispatcher component with the interested components is data, and the coupling of the dispatcher component will be (n/2).w_1. The coupling type of independent components (n/2) is data, so coupling of this style is obtained from $\sqrt{((n/2).w_1)^2+(n/2).w_1 2}$.

F. Client/server style. The coupling of the client with the server is data type. Supposing that, in average, the coupling of each server component is f and, since some server components are in transaction and usually the last component is related to repository, thus about r % of the components have just one connection with the repository. So, coupling of this style is obtained from $\sqrt{w_1^2+(1-r)n(f.w_1)^2+rnw_3^2}$.

G. Broker style. Coupling of all components is data type. Considering these facts: (1) the client component is related to the client side proxy, (2) the client is related to the broker in order to be informed of different services of the server, (3) the server side proxy is coupled with the broker, (4) the broker is coupled with the server side proxy and (5) the broker is coupled with the server for being informed of different type of services of the server, and also considering the similarity of the coupling of the server components to client/server style, the style coupling is obtained from $\sqrt{8w_1^2+(1-r)n(f.w_1)^2+r.n.w_3^2}$.

H. Object oriented style. In this style, the type of coupling is data. A case study done by Yu and Ramaswamy [28] on components dependency showed that 83% of the couplings between classes are of parameter (data) type. Coupling of each class with other classes is considered as f_o. So style coupling is obtained from $f_o\sqrt{n_o}\cdot w_1$.

Column 2 of Table 3 shows the coupling formulas of SASs. The third column shows the coupling value obtained by replacing the weight of coupling type based on Table 1.

Table 3. Coupling Formulas of SASs

Symbol	Coupling Formula	Coupling Value
RPS	$\sqrt{n}.w_3$	$3\sqrt{n}$
BKB	$\sqrt{(n.w_1+w_3)^2+n.w_3^2}$	$\sqrt{(n+3)^2+9n}$
P/F	$\sqrt{n-1}\ w_2$	$2\sqrt{n-1}$
LYD	$\sqrt{4(n-2).W_1^2+2.W_1^2}$	$\sqrt{4n-6}$
I/I	$\sqrt{((n/2).w_1)^2+(n/2).w_1 2}$	$\sqrt{(n/2)^2+(n/2)}$
C/S	$\sqrt{w_1^2+(1-r)n(f.w_1)^2+rnw_3^2}$	$\sqrt{1+(1-r)n.f^2+9.r.n}$
BRK	$\sqrt{8w_1^2+(1-r)n(f.w_1)^2+r.n.w_3^2}$	$\sqrt{8+(1-r)n.f^2+9.r.n}$
OO	$f_o\sqrt{n_o}\cdot w_1$	$f_o\sqrt{n_o}$

The coupling value of classes in object-oriented style (f_o) is related to the designing manner of the past software systems. This is true for the coupling value of server components (f) in the broker and client/server styles as well. Therefore, software designers determine the average value of coupling (i.e. f and f_o) by referring to the previous software design records. For displaying the relationship between coupling value and software size, it is necessary that first the values of f , r and f_o parameters are determined. Thus, documents of software design projects of a large and valid software company in Iran is investigated. Accordingly, after computations, the values of these parameters become f=1.65 and f_o=1.5 and r=0.2. By setting the parameters of f, f_o and r to the designated formulas and parameters n and n_o, the coupling value of SASs is computed considering the software size (number of components), and its diagram is shown in figure 1. According to this diagram, the coupling value of SASs is increased by increasing of the software size.

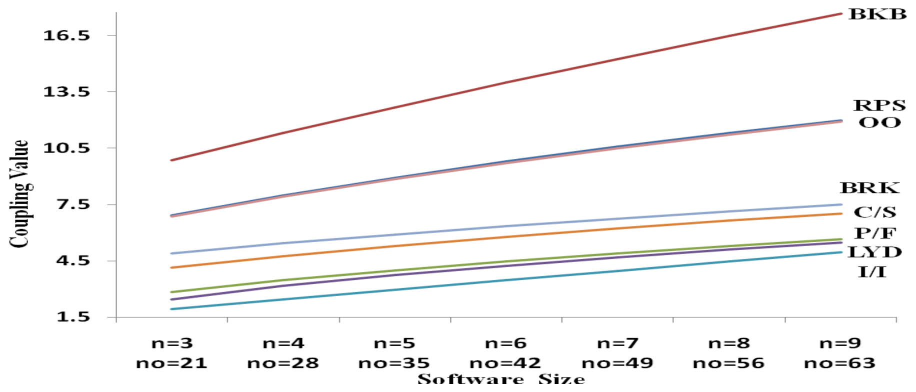

Figure 1. Coupling value of SASs based on the size of software

3.2. Measuring the Complexity of SASs

In this section, the complexity formula of every SASs is computed using (3) to (6).

A. Repository style. In this style, all components read from the data repository and modify it. Thus, both their fan-in and fan-out is equal to 1. Therefore, the total fan-out of each component, considering the writing in the repository and invoking the repository for this writing, is 3. Thus the complexity of independent components is 9 and the complexity of style is obtained from $9\sqrt{n}$.

B. Blackboard style. The fan-in of the control component is 1 (for examining the status of the blackboard) and its fan-out is 2 (for invoking the blackboard for reading its status and invoking the knowledge resources). The fan-in of knowledge resources is 2 (for invoking by the control component and reading from the blackboard) and its fan-out is 3 (for invocation of the blackboard for reading and writing into the blackboard). So the complexity of the control component is 2^2, the complexity of each of the knowledge resource is 36, and complexity of style is obtained from $\sqrt{4^2+36^2 n}$.

C. Pipe and filter style. The first filter (component) has no input and the last filter does not have any output. Thus, their complexity is 0. The other filters have one input and one output. So the complexity of style is obtained from. $\sqrt{n-2}$

D. Layered style. In this style, the relation of lower layer to upper layer is response to the request of upper layer, so in computing of layer's fan-out, this relation is ignored, i.e. only upper layer invokes lower layer. Thus, each layer has fan-in and fan-out equal to 1. None of the layers does not invoke first layer and the last layer invokes no layer. So their complexity is 0 and the complexity of style is obtained from $\sqrt{n-2}$.

E. Implicit invocation style. With the occurrence of an event, the dispatcher component invokes the interested components. Therefore, the fan-in of dispatcher component is 1 (for occurrence of the event that led to the invoking of the interested component by the dispatcher component) and its fan-out is 1 (for invocation of the interested component, when an event occurs). Therefore, its complexity is 1. The complexity of event publisher due to lack of fan-in and the complexity of interested components due to lack of fan-out is 0. Therefore, the complexity of style is 1.

F. Client/server style. The client component invokes a procedure from the server, so fan-in of the server and fan-out of the client is equal to 1. Since the client is not invoked by the components and has no direct access to the repository, its fan-in is equal to 0 and its complexity is 0. The number of fan-ins and fan-outs of the server components, in average, is considered as f. So the complexity of each server component is f ^{4} and the complexity of style is $f^4\sqrt{n}$ **.**

G. Broker style. The client component gets informed of the services of the server through the method interface of the server that has been offered to the broker component, so both fan-in and fan-out of the server becomes 1. In addition, fan-in of the broker becomes 1 due to accessing the interface of the server services. The client invokes the client side proxy, thus its fan-out becomes 1 as well. The client side proxy sends a request to the broker component, therefore, both its fan-in and fan-out become 1. The broker component sends the request to the server side proxy. On the other hand, the broker invokes the server to get informed of the interface of the server services. Therefore, both fan-in and fan-out of the broker become 2. The server side proxy has the fan-in and fan-out equal to those of the client side proxy too. The complexity of server components is considered similar to that of the client/server style, thus style complexity is obtained from $\sqrt{274+f^8 n}$.

H. Object-Oriented style. If, in average, the number of fan-in and fan-out of each class is considered as f_o, then the complexity of each class becomes f_o^4 and the complexity of style is $f_o^4\sqrt{n}_o$.

Table 4 shows the complexity formulas of SASs. Values of f and f_o are considered as similar to those in the Section 3.1.

Table 4. Complexity Formulas of SASs

Symbol	**Complexity Formula**
RPS	$9\sqrt{n}$
BKB	$\sqrt{4^2+36^2 n}$
P/F	$\sqrt{n-2}$
LYD	$\sqrt{n-2}$
I/I	1
C/S	$f^4\sqrt{n}$
BRK	$\sqrt{274+f^8 n}$
OO	$f_o^4\sqrt{n}_o$

By setting the parameters n and no, the complexity value of SASs is computed considering the software size and its diagram is shown in figure 2. According to this diagram, the complexity value of most SASs is increased by increasing of the software size.

3.3. Measuring the Cohesion of SASs

In this section, the cohesion formula of every SAS is computed using (7) to (8).

A. Repository style. Each component processes the same set of data, so their cohesion type is communicational. The repository component performs various functions on the data, and in each calling, one of the functions is performed. So its cohesion type is logical, and the cohesion of style is $\sqrt{n C_2^2 + C_1^2}$.

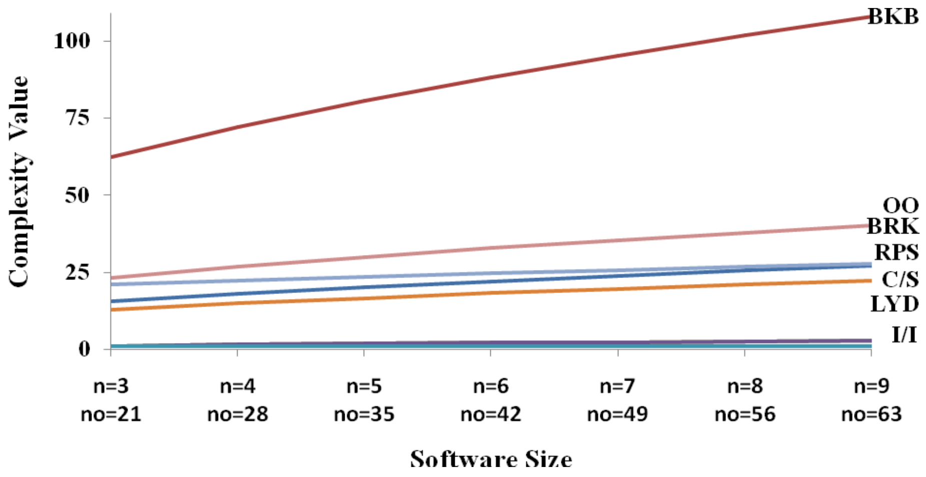

Figure 2. Complexity value of SASs based on size of software

B. Blackboard style. Each knowledge resource processes the same set of data, so their cohesion type is communicational. The control component invokes the knowledge resources based on the status of the blackboard. Therefore, its cohesion type is logical. The blackboard component performs various functions and, in each invocation, one of these functions is performed. So, its cohesion type is logical, and the cohesion of style is $\sqrt{n C_2^2 + 2 C_1^2}$.

C. Pipe and filter style. Each filter processes the same set of data, so its cohesion type is communicational and the cohesion of style is $\sqrt{n} . C_2$.

D. Layered style. Each layer contains some components; regarding the invoking of upper layer, one of components of the lower layer is performed, so the cohesion type of each layer is logical and the cohesion of style is $\sqrt{n} . C_1$.

E. Implicit invocation style. Since the components are publisher or interested in the event, their cohesion type is communicational. The dispatcher component performs various functions and, in each invocation, one of them is performed. Thus, its cohesion type is logical and the cohesion of style is $\sqrt{C_1^2 + nC_2^2}$.

F. Client/Server style. The server provides various services for the client by its components, and in each invocation, one or some of the server components are performed so that each one works on the same data. Accordingly, their cohesion type is communicational. The client component performs a specific function, so its cohesion type is functional. The repository component performs various functions and in each calling, one of them is performed. So its cohesion type is logical and the cohesion of style is $\sqrt{C_1^2 + nC_2^2 + C_3^2}$.

G. Broker style. The client side proxy, server side proxy, broker and server components perform various functions and in each invocation, just one of the functions is performed, so their cohesion type is logical. The client component performs a specific function, thus its cohesion type is functional. The repository component performs various functions and in each calling, one of them is performed. Therefore, its cohesion type is logical. Cohesion of the server components is considered similar to that of the client/server style. Thus, the cohesion of style is $\sqrt{4C_1^2 + nC_2^2 + C_3^2}$.

H. Object-Oriented style. The classes in this style define the data of an entity and its related functions, so, the cohesion type of each class is communicational and the cohesion of style is $\sqrt{n_o}.C_2$.

Column 2 of Table 5 represents the cohesion formulas of SASs. The third column shows the cohesion value obtained by replacing the weight of cohesion type based on Table 2. By setting the parameters n and no, the cohesion value of SASs is computed considering the software size and its diagram is shown in figure 3. According to this diagram, the cohesion value of SASs is increased by increasing of the software size and the amount of increase is higher in the object-oriented style relative to the other styles.

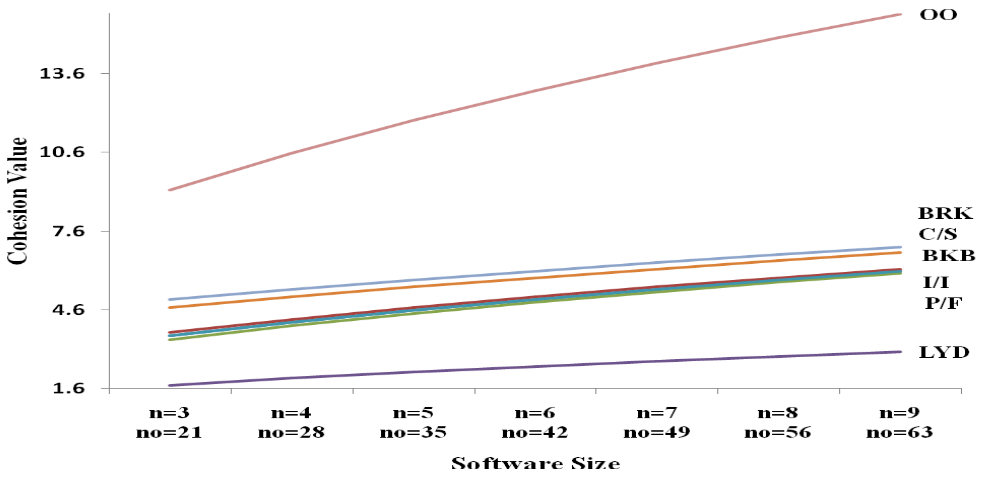

Figure 3. Cohesion value of SASs based on the software size

Table 5. Cohesion Formulas of SASs

Symbol	Cohesion Formula	Cohesion Value
RPS	$\sqrt{n C_2^2 + C_1^2}$	$\sqrt{4n+1}$
BKB	$\sqrt{n.C_2^2 + 2C_1^2}$	$\sqrt{4n+2}$
P/F	$\sqrt{n}.C_2$	$2\sqrt{n}$
LYD	$\sqrt{n}.C_1$	$\sqrt{n}$
I/I	$\sqrt{C_1^2 + nC_2^2}$	$\sqrt{1+4n}$
C/S	$\sqrt{C_1^2 + nC_2^2 + C_3^2}$	$\sqrt{10+4n}$
BRK	$\sqrt{4C_1^2 + nC_2^2 + C_3^2}$	$\sqrt{13+4n}$
OO	$\sqrt{n_o}.C_2$	$2\sqrt{n_o}$

4. Computation of The Rank of SASs

In this section, the ranking of SASs is performed based on the results of measurement coupling, complexity and cohesion of SASs using AHP method.

4.1. Organizing Ranking Problem of SASs

In SASs ranking problem, aim is in the first level, metrics are in the second level and SASs are in the third levels of the structure.

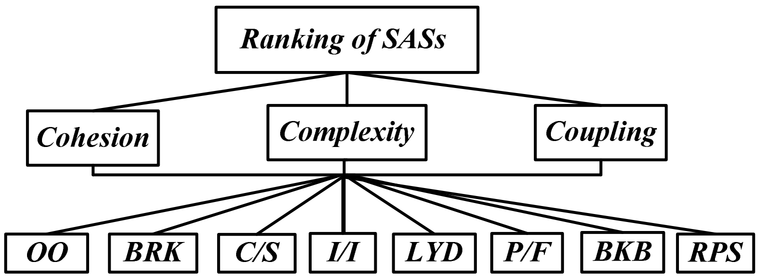

Figure 4. Hierarchical structure of SASs ranking

4.2. Computation of Priority of Metrics and the Relative Rank of SASs

In this stage, comparison matrix of the metrics and comparison matrices of SASs for the metrics are formed. The complexity and cohesion values of a component do not affect on the other components of SAS. However, the coupling value of a component affects the related components. Accordingly and due to the emphasis of researches on the importance of coupling [13], [15] the preference of coupling metric is considered more important than (1.6) the other metrics, and the preferences of other metrics are considered equal. Then the relative priority of metrics is computed by AHP method, the relative priority of coupling becomes 0.444 and that of the other metrics become 0.278.

To determine the relative rank of SASs for each metric, comparison matrices of SASs for each metric is formed. To set cell (i, j) of the comparison matrix of metric k, for the style x in row i with the style y in column j, if there is a direct relation between the metric k and maintainability, the ratio of the metric value of style x to the metric value of style y is set to cell (i,j), otherwise the inverse of the ratio is set to cell(i,j). After setting of the comparison matrices based on the described procedure, the relative rank of SASs for each metric is computed by AHP method. Investigation of the consistency using the Expertchoice tool, tool of AHP method, showed that consistency index is zero, so there is no inconsistency between the comparisons.

4.3. Computing the Final Rank of SASs

The final rank of SASs is computed regarding the priority of metrics and the relative ranks of SASs. Table 6 shows the final rank of SASs. Based on the values of this Table, the Implicit/ Invocation (I/I), Pipe and Filter (P/F), and Layered (LYD) styles provide the highest support for maintainability, respectively.

Figure 5 shows the changes in maintainability value of SASs based on the changes of software size. With the increasing of software size, the rank of some styles such as Pipe and Filter(P/F) and Layered (LYD) are decreased, and the rank of some styles such as Implicit Invocation(I/I) are increased while the rank of some styles such as Blackboard(BKB) are not changed considerably.

Table 6. Rank of SASs from the maintainability viewpoint

Symbol	**n=3**	**n=4**	**n=5**	**n=6**	**n=7**	**n=8**	**n=9**
	no=21	**no=28**	**no=35**	**no=42**	**no=49**	**no=56**	**no=63**
RPS	64	67	69	69	70	70	70
BKB	52	54	54	55	55	55	55
P/F	187	176	170	166	163	160	158
LYD	185	169	161	155	151	149	146
I/I	223	238	246	251	255	257	260
C/S	95	97	97	98	99	99	99
BRK	87	89	91	92	94	94	95
OO	107	110	112	114	115	116	116

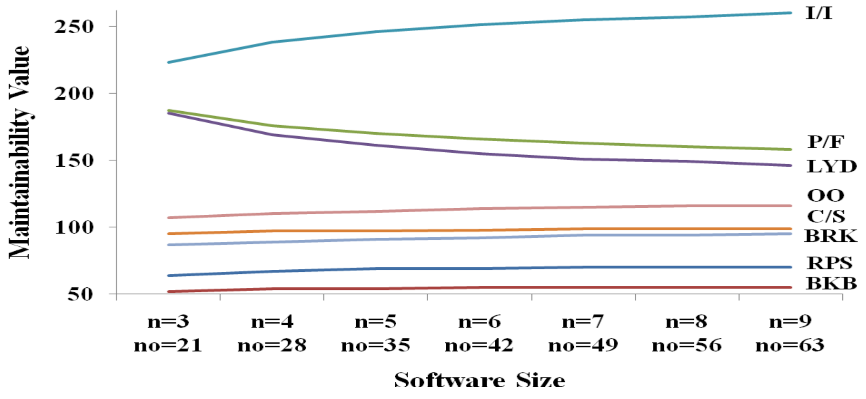

Figure 5. Maintainability value of SASs based on the changes of software size

Figure 6 shows the diagram of styles ranks based on the relative priority of metrics. It is known as sensitivity analysis diagram, which is drawn by Expertchoice. In this diagram, the vertical lines show the relative priority of metrics and the horizontal lines show the rank of SASs based on the metrics. The final rank of SASs is determined by the "OVERALL" label based on the vertical line (figure 6). The coupling metric accords with the y-axes and after that are complexity, cohesion and combination of the three metrics.

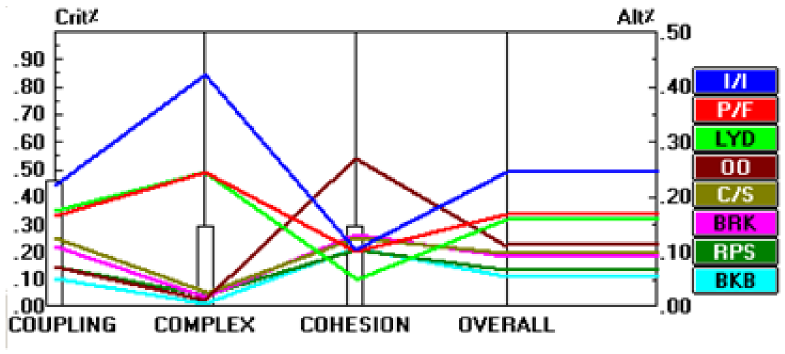

Figure 6. Diagram of styles rank regarding the relative priority of metrics

4.4. Analyzing the Rank of SAS

Here, by changing the values of some parameters, the effects of these changes on the rank of SASs are investigated.

- For the values of coupling types (Section 2.A), other values were used besides the values mentioned in table 1 (for twelve values in the ranges $1 \le w1 \le 1.5$, $1.5 \le w2 \le 2.5$ and $2.5 \le w3 \le 3.5$), but they did not lead to any changes in the rank position of the SASs' maintainability.
- For the values of cohesion types (Section 2.C), other values were used besides the values mentioned in table 2 (for twelve values in the ranges $1 \le c1 \le 1.5$, $1.5 \le c2 \le 2.5$ and $3 \le c3 \le 3.5$), but they did not lead to any changes in the rank position of the SASs' maintainability.
- By changing the f parameter (coupling of the server components in Section 3.A) in the range of $1.65 \le f \le 2.8$ at the Client/Server (C/S) style, the change in the rank position of this style was checked. It was found that only for $f \geq 2$, the rank position of this style is placed after the Broker (BRK) style and no other change in the rank position of other styles was seen.
- For determining the relative priority of metrics (In Section 4.B), in addition to 1.6 (the relative priority of coupling metric compared to that of the other metric), the ten values in the range of 1.3 to 2.2 were used. The results showed no changes in the rank position of the styles from maintainability viewpoint.

5. CONCLUSION

In this study, a model was offered to analyze the impact of SASs on software maintainability according to the measurement-based evaluation of SASs. In this model, first, the formulas were presented to compute the coupling, complexity and cohesion values of each SAS. Next, the coupling, complexity and cohesion values of SASs were computed quantitatively using the presented formulas. Then, the relative rank of each SAS was determined regarding the coupling, complexity and cohesion values of SASs. Afterward, the priority of metrics was determined. Subsequently, the final rank of SASs maintainability was determined using AHP method.

The analyses done showed that our proposed method had stability regarding the value of coupling types, different values of f parameter, value of cohesion types and preference of coupling metric to the other metrics.

Since the evaluation of this paper is based on measurement as compared to the method used in [9], which uses scenario-based evaluation and the quality of its results is dependent on the used scenarios and also on the extensive expert participation, the results of our proposed model is more precise, more reliable and more analyzable.

The proposed method gives formulas to determine the values of 1) coupling, 2) complexity and 3) cohesion of each SAS, while this has not been done in previous methods.

As compared to [4], [8], both the proposed method and the method used in [9] give the quantitative results about the maintainability of SASs that is basis of the systematic recommendation and selection of SAS.

Finally, only the proposed method examines the effect of the software size on the maintainability rank of SASs.

The methods given [6], [7], [11] use the mathematic model-based evaluation and the method used in [10] uses the simulation-based evaluation. These methods verify specific features such as consistency and satisfaction of some properties by SASs that are different from the quality attributes required in this paper. The above points and table 8 clearly show the position of the proposed method as compared to the methods of [4], [8] and [9].

It is worth noting that the ranking of SASs based on our proposed method is consistent with the priorities of SASs from the viewpoint of maintainability in the methods used in [2], [12], which are based on experimental studies.

Table 7. Comparison of the proposed method with the related methods

Method / **Criteria**	**Proposed Method**	**Method [4]**	**Method [8]**	**Method [9]**
Base	Measurement	Tree	Unsystematic	Scenario
Offering the Quantitative Results about the Maintainability of SASs	●			●
Total SASs that were Investigated	8		6	8
Considering the Effect of Software Size on the Rank of SASs	●			

REFERENCES

[1] Len Bass, Paul Clements. & Rick Kazman(2003) Software Architecture in Practice (2nd Edition), Addison-Wesley, p 89.

[2] F. Buschmann, R. Meunier, H. Rohnert, P. Sornmerlad, & M. Stal,(1996) Pattern-Oriented Software Architecture- A system of Patterns" John Wiley & Sons, p. 394.

[3] C. Seo, G. Edwards, S. Malek, & N. Medvidovic,(2009) "A Framework for Estimating the Impact of a Distributed Software System's Architectural Style on Its Energy Consumption", 7th Working IEEE/IFIP Conf. on Software Architecture, pp. 277-280.

[4] B. Harrison, & P. Avgeriou,(2007) "Leveraging Architecture Patterns to Satisfy Quality Attributes", 1st European Conf. on Software Architecture, Springer, pp. 263-270.

[5] P. Avgeriou P, & U. Zdun, (2005) "Architectural Patterns Revisited: A Pattern Language", Proc. of 10th European Conf. on Pattern Languages of Programs, pp.1-39.

[6] J.S Kim, and D. Garlan, (2006) "Analyzing Architectural Styles with alloy", Proc. of the ISSTA 2006 workshop on Role of Software Architecture for Testing and Analysis, pp. 70-80.

[7] R. Bruni, A. Bucchiarone, A. Gnesi, D. Hirsch, & A.L. Lafuente, (2008) "Graph-based Design and Analysis of Dynamic Software Architectures", LNCS 5065, pp. 37–56,.

[8] H. Reza, & E. Grant, (2005) "Quality-Oriented Software Architecture", the IEEE Int. Conf on Information Technology, pp. 140 – 145.

[9] Gholamreza Shahmohammadi, & Saeed Jalili, (2009) "Scenario-Based Quantitative Evaluation of Software Architecture Style from Maintainability Viewpoint", 14 th Annual of CSI Computer Conference (CSICC 2009), Iran, Amirkabir University.

[10] H. Grahn, & J. Bosch, (1998) "Some Initial Performance Characteristics of Three Architectural Styles", Proc. of Int. Workshop on Software and Performance.

[11] D. Garlan, & S. Khersonsky, (2000) "Model Checking Implicit Invocation Systems", 10th Int. Workshop On Software Specification and Design.

[12] M. Shaw & D. Garlan, (1996) Software Architecture: Perspectives Discipline on an Emerging Discipline, Prentice Hall.

[13] L. Briand, S. Morasca, & V. Basili, (1996) "Property Based Software Engineering Measurement", IEEE Trans on Software Eng., vol. 22, no. 1, pp. 68-86.

[14] L. Briand, J. Wust, & H. Lounis, (1999) "Using Coupling Measurement for Impact Analysis in Object-Oriented Systems", IEEE Int. Conf. on Software Maintenance.

[15] S.L. Pfleeger, & J.M. Atlee, (2006) "Software Engineering, Theory and Practice", 3rd Edition, Prentice Hall.

[16] P. Yu, T. Systa, & H. Muller, (2002) "Predicting Fault Proneness using OO Metrics. An Industrial Case Study," 6th European Conf. on Software Maintenance and Reengineering, pp.99 – 107.

[17] M. Alshayeb, and L. Wei, (2003) "An Empirical Validation of Object-Oriented Metrics in Two Different Iterative Software Processes," IEEE Trans on Software Engineering, vol. 29 (11), pp. 1043 – 1049.

[18] F. Bachmann, L. Bass, M. Klein, M. & C. Shelton, (2005) "Designing Software Architectures to Achieve Quality Attribute Requirements", IEE Proc. of Software, Vol. 152, No 4,pp. 153- 165.

[19] C.L. Hwang, K. Yoon, (1981) "Multiple Attribute-Decision Making", Springer-Verlag.

[20] T. L. Saaty, & L. G. Vargas, (2001) "Models, Methods, Concepts & Applications of the Analytic Hierarchy Process", Kluwer Academic Publisher.

[21] L. Bass, P. Clements, & R. Kazman, (1998) Software Architecture in Practice, Addison-Wesley, p. 17.

[22] ISO, (2001), International Organization for Standardization, "ISO 9126-1:2001, Software Engineering – Product quality, Part 1: Quality model".

[23] E. Yourdon, & L. Constantine, (1978) Structured Design, Englewood Cliff, NJ, prentice Hall.

[24] N. Fenton, & A. Melton, (1990) "Deriving Structurally Based Software Measures", Journal of Systems and Software 12(3), pp. 177-187.

[25] M. J. Shepperd, & D.C. Ince, (1990) "The use of metrics in the early detection of design errors", Proc.of the European Software Engineering Conf, pp.67-85.

[26] NE. Fenton, & SL. Pfleeger, (1997) "Software Metrics: A Rigorous and Practical Approach", (2nd Edition), International Thomson Computer PRESS.

[27] S. Chidamber, & C. Kemerer, (1994) "A Metrics Suite for Object Oriented Design", IEEE Trans on Software Engineering, vol. 20, pp. 476-493.

[28] L. Yu, & S. Ramaswamy,(2007) "Component Dependency in Object-Oriented Software", Journal of Computer Science and Technology, 22(3), pp. 379-386.

[29] Lee, S.hyun. & Kim Mi Na, (2008) "This is my paper", ABC Transactions on ECE, Vol. 10, No. 5, pp120-122.

AUTHORS

Gholamreza Shahmohammadi received his Ph.D. degree from Tarbiat Modares University (TMU, Tehran, Iran) in 2009 and his M.Sc. degree in Computer Engineering from TMU in 2001. Since 2010, he has been Assistant Professor at the Olum Entezami University-Amin(Tehran, Iran). His main research interests are software engineering, quantitative evaluation of software architecture, software metrics and software cost estimation.

E-mail: Shah_mohammadi@yahoo.co.uk