RELIABILITY EVALUATION OF SOFTWARE ARCHITECTURE STYLES

Gholamreza Shahmohammadi

Department of Information Technology,
Olum Entazami Amin University, Tehran, Iran
Shahmohamadi@yahoo.co.uk

ABSTRACT

In process of software architecture design, different decisions with system-wide impacts are made. An important decision of design stage is the selection of appropriate software architecture style. Since quantitative impacts of styles on quality attributes have not been studied yet, their application is not systematic. Since Reliability is one of the essential quality requirements of software systems, especially for life critical ones, one of the main criteria in choosing architecture style of these systems is high reliability. The goal of this study is to quantify the impact of architecture styles on software reliability that is desired quality of life critical software. We evaluate styles through reliability block diagram method. First, the reliability equation of each architectural style was computed using of Reliability block diagram approach. Then, reliability rank of architectural styles is computed by setting of the number of effective components in a transaction parameter in reliability equation of architectural styles. The main innovation of this article is quantification of impact of styles on software reliability that is essential for style selection.

KEYWORDS

Software Architecture, Software Architecture Style Evaluation, Reliability block diagram

1. INTRODUCTION

Software systems are increasingly entering consumers' everyday life. To satisfy the consumers' requirements, these systems must demonstrate high reliability and availability. Thus, they must function correctly and without interruption [1]. The software architecture design stage is the first stage of software development in which it is possible to evaluate how well the quality requirements are being met [1]. In the process of architecture design, different decisions are made that have system-wide impact [2]. Architectural decisions made early in the design process are a critical factor in the successful development of system. In particular, the selection of an appropriate software architecture style (SAS) has a significant impact on various system quality attributes [3]. Functionality may be achieved using any of a number of possible structures [4], so SASs are selected based on amount of their support from quality attributes. Styles present models for solving 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 [5]. Since quantitative impacts of SASs on quality attributes have not been studied yet [6], their applications are not systematic [7]. In other words, present use of styles in design is based on intuition of software developers.

Software reliability is widely recognized as one of the most important aspects of software quality [8]. Thus, quantification of impact of SASs on software reliability plays an important role in selecting SASs. This study is a step towards quantification of influence of SASs on quality attributes.

In [9], a method is shown to map an architectural style, expressed formally in an architectural description language, into a relational model that can be checked for various style properties such as consistency. In [3], the impact of a distributed software system's architectural style on the system's energy consumption has been estimated. In [10], a method for specifying the relationship between six SASs and quality attributes such as reliability has been proposed. The relationship between the quality attributes, design principles and some SASs has been specified using a tree-based framework. In [6], impacts of SASs on quality attributes are determined based on the description of style in [11]. [6] and [10] are not able to determine the amount of style support from quality attributes and do not offer quantitative results about their reliability and are not precise.

Different methods for evaluating software reliability are presented [12, 13, and 14]. Considering available information about SASs, only some of those methods like reliability block diagram (RBD) [14] can be used for evaluation of SASs reliability.

In this study, according to the concept of transaction, SASs are evaluated based on reliability block diagram approach and reliability equation of SASs in this approach is determined. Then reliability rank of SASs are determined by setting the number of effective components in a transaction (NECT), and impact of different NECT on SASs rank is investigated. Thus, quantitative impact of SASs on software reliability is determined.

The RBD method presents a block representation of software components and their reliability status, and is a suitable method for determining the reliability of SASs and the estimation of their reliability.

In this work, eight styles: 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) are evaluated from reliability viewpoint.

The paper is organized as follows: in section 2 SASs, in section 3 software reliability and their evaluation methods are discussed. In section 4 reliability evaluations of SASs and in section 5 ranking of SASs is described. In section 6 conclusions are explained.

2. SASS

SASs present models for solving 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 [11]. Shaw and her colleague [5] introduce seven SASs. Buschmann et al [11] have also described the pattern in different levels.

Since the reliability of SASs are evaluated in this study, regarding effective components in transactions, transaction definition and eight SASs introduced briefly. It should be noted that a transaction is a set of operations that consists a logical unit of the job [15]. Since the evaluation is performed at the level of architectural styles, the transaction is considered in term of effective components in the transaction.

Repository style (RPS). In this style, there are two types of components: a central storage and a set of components that store, retrieve and update information on the repository [5].

Blackboard style (BKB). The components of this style are Blackboard, experts (knowledge resources), and the control. The control component in a loop, checks the blackboard status, evaluates knowledge resource, and activates one of them for the execution [5].

Pipe and filter (P/F). This style is composed of a set of computational components. Each component acts as a filter and has a number of inputs and outputs. The output of each component is the input of the next component [5].

Layered style (LYD). In this style, the emphasis is on different abstraction level in the software. The layered style organized hierarchically. Each layer provides a service for its above layer and uses its lower layer [5].

Implicit Invocation (I/I). Implicit invocation style is an event-driven style based on broadcast concepts and announces the occurrence of the event instead of directly invoking a function. Interested components relate a function to an event. With the occurrence of an event, software invokes all registered functions [5]. Components of this style are: (1) event publishers, (2) components that are interested in events, and (3) dispatcher that invokes interested components in response to an event occurrence.

Client/server(C/S). The components of this style are clients and servers. Clients should be aware of the name and services presented by servers [5].

Broker style (BRK). Client, servers, broker, client side proxy and server side proxy are the components of this style. Broker is responsible for coordinating the relationship between clients and servers. Servers register themselves with the broker, and make their services available to clients through method interfaces. Clients access services of servers by sending requests via the broker. Locating appropriate servers, forwarding the request to it and return the results to the client are the responsibilities of the broker [11].

Object-oriented (OO). In this style, data presentations and the related operations encapsulated in an object. Objects are the components of this style and they interact through invoking the functions [5].

3. SOFTWARE RELIABILITY AND THEIR EVALUATION METHODS

The main objective of software is to offer services desired according to the predetermined quality level. The quality of software has a direct relationship with software architecture. Software often redesigned not because they are functionally deficient, but because they are difficult to maintain, port, or scale[16].

According to IEEE standard 610-12[17] software architecture should provide two types of quality requirements: developmental and operational. Developmental quality Requirements such as maintainability and reusability are important in software development in future. Operational quality requirements such as performance and reliability are important for software users and if the software lacks them, will not be used.

Reliability is a set of sub-characteristics that determines the capability of the software to maintain performance under stated conditions for a stated time period [18]. In other words, reliability indicates a time during which the software is available for use. Reliability sub-characteristics are as follows [18]:

- Maturity: the capability of the software product to avoid failures, as a result of faults in the software. The less the frequency of failure, the higher the software maturity will be.
- Fault tolerance: the ability to maintain a specified level of performance in case of software fault.
- Recoverability: Capability to re-establish the level of performance, Capability to recover the data, and the time and effort needed for it.

The analysis of reliability sub-characteristics indicates the following points:

- The frequency of software failure is directly related to the number of critical components of software architecture, because the more the number of software architecture critical components, the higher the potential possibility of software failure will be.
- Software fault tolerance has a reverse relationship with the number of SAS critical components, because the more the number of SAS critical components, the lower the software fault tolerance will be.
- Recoverability has a reverse relationship with the number of SAS critical components, because principally recoverability is discussed about components by failure of which the performance of software decreases substantially.

Considering the above-mentioned, RBD approach that do reliability evaluation based on software components and their interactions, is suitable method for evaluation of reliability at the level of SAS.

3.1. Software Reliability Evaluation Methods

The software Reliability assessment methods have been classified into quantitative and qualitative methods [1]. There are different types of quantitative methods; some of them are usable before software implementation and some after. Measurement based methods which focuses on the failure of the system are usable before and after software implementation.

RBD method has been presented as a method based on software structure in order to evaluate reliability of software systems. Tripathi et al. [19] have used RBD to estimate reliability of complex software systems from a hierarchy of modules. Leblanc et al. [20] presented a model based on RBD for indicating real world issues and an algorithm for analysis of this model at the early stage of software development. So this method can be used to evaluate SAS reliability.

3.1.1. Reliability Block Diagram (RBD) Method

RBD is a block representation of software components in order to investigate the reliability of components [14]. Since in each architectural style the constituent components the architecture and their interactions are known, RBD method is suitable for evaluating SASs. RBD method, determines SASs reliability based on their structure. SASs reliability prediction is done through investigating the reliability of components. In order to create an RBD the configuration of software architecture components should be determined first. Figure 1 represents a sequential configuration of components. In a sequential configuration, a failure of any component results in failure for the software. The reliability of software with sequential configuration equals the probability that all the components of the architecture of that software succeed. In this case, the reliability of software is computed by Eq. (1). Where R_s is software reliability, X_i is event of

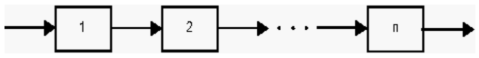

Figure 1. Block diagram of sequential configuration [14]

$$R_s = P(X_1 \cap X_2 \cap \dots \cap X_n) = P(X_1)\, P(X_2 | X_1)\, P(X_3 | X_1 X_2) \dots P(X_n | X_1 X_2 \dots X_{n-1}) \quad (1)$$

component i being operational, and P(Xi) is the probability that component i is operational. If components are considered independent then Eq. (1) becomes Eq. (2). In this Eq., Ri is the reliability of component i. In a

$$R_s = \prod_{i=1}^{n} P(X_i) = \prod_{i=1}^{n} R_i \quad (2)$$

Sequential configuration, the component with the smallest reliability has the biggest effect on software reliability. As the number of components increases, the software's reliability decreases. Similarly, RBD method can be used for computing software reliability with parallel and sequential-parallel configuration.

4. Reliability Evaluation of SASs

In section 4.1, RBD method was presented to compute SASs reliability.

Although methods have been offered to predict software components reliability, since the evaluation is done at the architectural styles level, there is no enough information of SASs components. Thus, the reliability of each component is considered Ri.

4.1. Reliability Evaluation of SASs using RBD Method

In this section, SASs will be evaluated using RBD method.

Repository style (RPS). In this style, an independent component C_i interacts with the repository in a transaction. As a result, reliability of style is computed by Eq. (3), where R_{rps} is the reliability of the repository component, and R_i is the reliability of each of the independent components.

$$R = R_i . R_{rps} \tag{3}$$

Blackboard style (BKB). For a transaction in BKB, the control component (C_c) checks the blackboard component (C_{bkb}) status and selects a suitable knowledge resource (C_{kr}), then the knowledge resource interacts with the blackboard. Thus, the reliability of style is computed by Eq. (4), where R_c is the reliability of the control component, R_{bkb} is the reliability of the blackboard, and R_{kr} is the reliability of the knowledge resource.

$$R = R_c R_{bkb} R_{kr} \tag{4}$$

Pipe and Filter style (P/F). In this style, in order to perform a transaction all components should be active. Since in each transaction m components are effective, the reliability of P/F is computed by Eq. (5), where R_i is the reliability of each filter.

$$R = \prod_{i=1}^{m} R_i \tag{5}$$

Layered style (LYD). In this style in order to perform a transaction, all layers must be active. Thus, reliability of style is also computed by Eq. (5). R_i is the reliability of each layer.

Implicit invocation style (I/I). In this style, a transaction begins with occurrence of an event. Then the event dispatcher component (C_d) activates interested component (C_i). After the interested component finishes its activity, transaction ends. Thus, the reliability of this style is computed by Eq. (6), where R_d is the reliability of event dispatcher component and R_i is the reliability of component interested in the event.

$$R = R_d . R_i \tag{6}$$

Client/Server style (C/S). In this style, a transaction begins by sending the request of client to the server. After the request is processed by the server that usually needs interaction with repository component, the result is sent to the client. Since the server is a complex component, and consists of several components, the execution of the server is in fact the execution of m components. Thus, in a transaction, m+1 components, consist of repository and m components in the server, are each effective. Therefore, the reliability of this style is computed by Eq. (7), where R_{rps} is the reliability of the repository component and $R_1 \ldots R_m$ are the reliability of server components.

$$R = R_m \ldots R_2 R_1 R_{rps} \tag{7}$$

Broker style (BRK). In broker style, a transaction begins by sending the request of client. The broker component (C_{brk}), client side proxy (C_{csp}), server (C_s) and repository (C_{rps}) are effective in the transaction. Similar to client/server style, the execution of the server is in fact execution of m components. Thus, in a transaction, m+7 components are effective. so the reliability of this style is computed by Eq. (8), where R_{csp} is the reliability of client side proxy, R_{brk} is the reliability of the broker component, R_{ssp} is the reliability of the server side proxy, and R_{rps} is the reliability of the repository component.

In server component of client/server and broker styles, a fraction of m components is effective in transaction. Figure 2 shows the diagram of change in reliability rank of client/server style in terms of the number of server components effective in transaction for m =5. According to diagram, by increasing the number of these components the reliability rank of the style is decreased. This value is considered m/2 based on authors' experience in producing software systems. Software developers determine this parameter based on their former experience in software development.

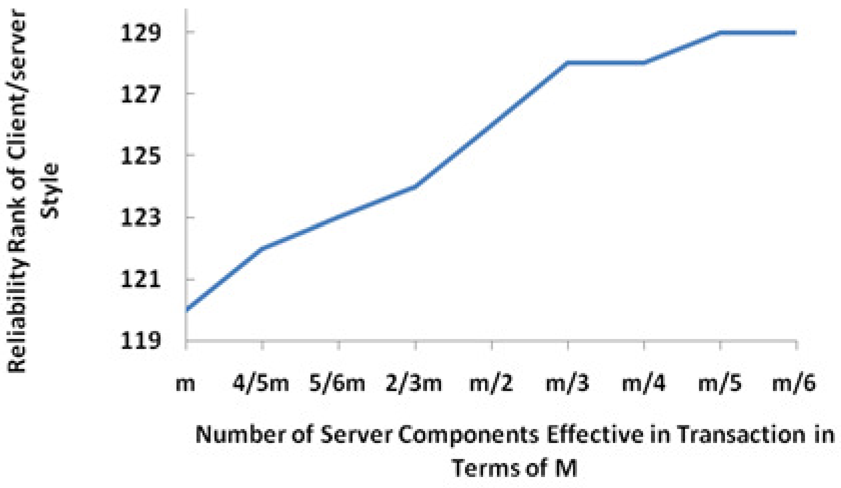

Figure 2. Diagram of change in reliability rank of client/server style in terms of the number of server components effective in transaction for m=5

$$R = R_{csp} R_{brk} R_{ssp} R_{m} \dots R_{2} R_{1} R_{rps} \tag{8}$$

Object-Oriented style (OO). In this style, the reliability of each use case that consists of k classes is computed by Eq. (9). An Object-oriented system includes a few use cases, and each use-case has its own execution path. By failure of one use case, the system continues its work with lower throughput. Thus, the configuration of this style is like figure 3, where each execution path indicates a use case. So, the reliability of the style is computed by Eq. (10), where p is the number of use cases, and R_u is the reliability of each use case. By substituting R_u into Eq. (10), the reliability of this style is computed by Eq. (12).

$$R_u = \prod_{i=1}^{k} R_i \tag{9}$$

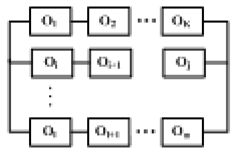

Figure 3. Block Diagram of various transactions

$$R = 1 - \prod_{u=1}^{p} (1 - R_u) \tag{10}$$

$$R = 1 - \prod_{u=1}^{p} (1 - \prod_{j=1}^{k} R_j) \tag{11}$$

In this equation, R_u reliability of each use case. By substitution of R_u from Eq. (12), Eq. (11) is obtained. Since some of the classes are common among use cases, the reliability of the style is calculated by Eq. (12) in which c is the number of common classes in use cases. In object oriented reliability equation, c indicates the number of common classes in use cases. Investigation of the reliability of parallel section ($1-\prod_{u=1}^{p}(1-\prod_{j=1}^{k}R_j)$) with different class number

$$\prod_{i=1}^{c}R_i(1-\prod_{u=1}^{p}(1-\prod_{j=1}^{k}R_j)) \tag{12}$$

showed that the reliability of this section is close to 1. So, reliability of the style is equal to that of sequential section ($\prod_{l=1}^{c}R_l$). So, reliability of this style is calculated by Eq. (13).

$$\prod_{l=1}^{c}R_l \tag{13}$$

Figure 4 shows the diagram of change in reliability of object oriented style in terms of percent common classes in use cases for n_o=15. According to diagram, by increasing this percent, reliability of the style is decreased. Based on their experience, the authors considered the percent of common classes in use cases to be 20% of all classes.

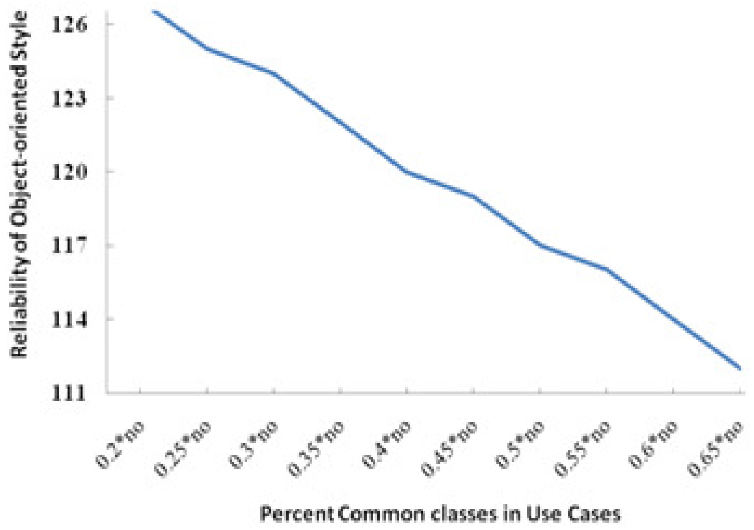

Figure.4. Diagram of change in reliability rank of client/server style in terms of the number of common classes of use cases for no=15

Software developers determine common class parameters in use cases and the number of effective server components in transaction based on their former experience in software development.

Table 1 indicates the reliability equation of SASs based on RBD method.

4.2. Reliability Evaluation of Large Systems

In some systems, not only an *SAS* is used as the main basis for system structuring, but also each of their components may use a specific *SAS*. Concerning determination of reliability of *SASs* in this study, it is also possible to evaluate and determine the reliability of such systems too.

Table 1. Reliability equation of SASs based on RBD method

Style	Symbol	Reliability
Repository	RPS	$R_i.R_{rps}$
Blackboard	BKB	$R_c R_{bkb} R_{kr}$
Pipe and Filter	P/F	$\prod_{i=1}^{m} R_i$
Layered	LYD	$\prod_{i=1}^{m} R_i$
Implicit invocation	I/I	$R_d R_i$
Client/Server	C/S	$R_{m/2}..R_2 R_1 R_{rps}$
Broker	BRK	$R_{csp} R_{brk} R_{ssp} R_{m/2} \ldots R_2 R_1 R_{rps}$
Object-oriented	OO	$\prod_{i=1}^{c} R_j$

5. RANKING OF SASS

In section 4, the reliability of *SASs* was computed. In the reliability Eq. of most *SASs*, a parameter 'm' exists which indicates *NECT*. So, in order to investigate the impact of *NECT*, the value of 'm' is considered as 2, 3, 4, 5, 6, and 7. The components of object-oriented style (i.e. classes) are more fine grain than components of other *SASs*. Thus, with the approximation of three classes in each component in a transaction in average, number of effective classes in a transaction, corresponding to the *NECT* in other *SASs*, will be considered as 6, 9, 12, 15, 18, and 21 the. We consider the reliability of each component/class 0.98.

For different values of m and based on the reliability Eq. of *SASs* in tables 1, the reliability of *SASs* is computed and is shown in tables 2.

Table 2. Reliability of SASs using RBD method

	m=2	m=3	m=4	m=5	m=6	m=7
Symbol	n_o =6	n_o =9	n_o =12	n_o =15	n_o =18	n_o =21
RPS	0.96	0.96	0.96	0.96	0.96	0.96
BKB	0.941	0.941	0.941	0.941	0.941	0.941
P/F	0.96	0.941	0.922	0.904	0.885	0.868
LYD	0.96	0.941	0.922	0.904	0.885	0.868
I/I	0.96	0.96	0.96	0.96	0.96	0.96
C/S	0.96	0.951	0.941	0.932	0.922	0.913
BRK	0.904	0.895	0.886	0.877	0.868	0.859
OO	0.976	0.964	0.953	0.941	0.93	0.919

The reliability rank of SASs has been computed by Eq. (13).

$$y_{ij} = \frac{x_{ij}}{\sum_{i=1}^{n} x_{ij}} \qquad (13)$$

X_{ij} is the reliability value of style in i-th row and j-th column. For more clarity, rank values have been shown by coefficient of 1000 in tables 3.

Table 3 shows changes in rank of *SASs* Reliability based on the changes of software size. With increasing software size, rank of some styles such as pipe and filter(*P/F*) and layered (*LYD*) are decreased, rank of some style such as Repository(RPS) and implicit invocation(*I/I*) are increased and rank of some style such as Client/server(C/S) has not changed considerably

Table 3. Reliability ranks of SASs using RBD method

	m=2	m=3	m=4	m=5	m=6	m=7
Symbol	n_o =6	n_o =9	n_o =12	n_o =15	n_o =18	n_o =21
RPS	126	127.1	128.3	129.4	130.6	131.7
BKB	123.4	124.5	125.6	126.7	127.9	129
P/F	126	124.6	123.2	121.9	120.5	119.1
LYD	126	124.6	123.2	121.9	120.5	119.1
I/I	126	127.1	128.3	129.4	130.6	131.7
C/S	126	125.9	125.7	125.6	125.4	125.3
BRK	118.6	118.5	118.4	118.2	118.1	117.9
OO	128.1	127.6	127.3	126.9	126.5	126.1

5.1. Analysis of Reliability Rank of Styles According to Reliability of Designed Components

In most *SASs*, the functionalities of some of components are deterministic without considering specific software. Other components are designed regarding the functionalities of the specific software. Thus, the components of each *SAS* are classified into two types: components with specific or constant functionalities, and designed components. For instance, in the broker style, functionalities of broker, client side proxy, and server side proxy components are constant. We consider the reliability of components with constant functionalities as 0.98. As it was mentioned in section 5.3, we consider the reliability of each component at the time of returning from calling of another component as 0.99.

In this section, by changing the reliability of designed components from 0.7 to 0.98, the impact of this change on the reliability of *SASs* for m=5 are studied. According to the graph shown in figure 5, by increasing the reliability of designed components, the rank-difference of the *SASs* reliability decrease. For R=0.7, the object-oriented style has the lowest and the repository style has the highest reliability rank. By increasing of the reliability of designed components, the trend of reliability rank in the repository, the blackboard and the implicit invocation style is decreasing, while the trend of reliability rank of other *SASs* is increasing.

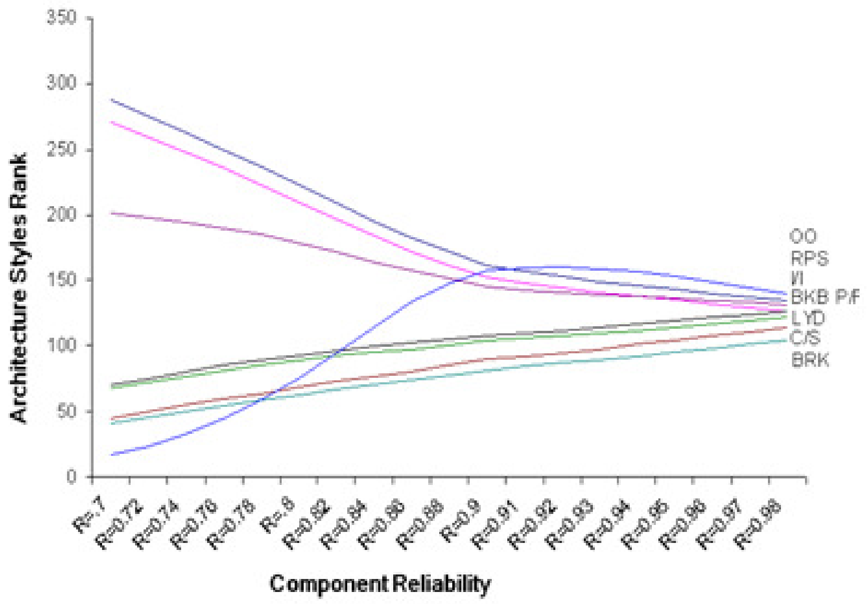

Figure 5. Reliability rank changing trend of *SASs* based on Change of the reliability of designed components

6. CONCLUSION

In this paper, due to the lack of study of quantitative *SASs* impact on quality attributes specifically reliability; *SASs* effect on the software reliability was analyzed. First, the reliability equation of each *SAS was* extracted using of *RBD* approach. Then, reliability rank of *SASs* is computed by setting of *NECT* parameter in reliability equation of *SASs* in ranges of 2 to 7. Thus, effect of different values of *NECT* on reliability rank of *SASs* is determined.

The most important innovation of this paper is quantification of software reliability at the level of *SASs* and at the stage of *SAS* selection essential to recommend and select *SASs*.

Author was evaluated *SASs* from maintainability viewpoint in [21] and this research extends previous research.

No other similar study was found dealing with evaluation, ranking and comparison of the reliability of *SASs*. In contrast to [6] and [10], the proposed model is based on architecture evaluation method. The proposed model offers: 1) equations to determine value of reliability of *SASs*, while [6] and [10] have not done so. In contrast to [6] and [10], the proposed approach offers quantitative results on *SASs* reliability essential to recommend and select *SASs* based on recognition of the quantitative effect of *SASs* on quality attributes. In addition, the proposed approach considered the impact of *NECT* on rank *of SASs* reliability. [9] is placed in mathematical model-based evaluation. This method verify features such as consistency and satisfy some features by *SASs* that are different from desired quality attributes of this paper.

REFERENCES

[1] Immonen A, Niemelä E,(2008) “Survey of reliability and availability prediction methods from the viewpoint of software architecture”, Journal of Software and Systems Modeling, Springer, Vol. 7, No.1, pp. 49-65.

[2] Jansen A G, Bosch J,(2005) “Software Architecture as a set of Architectural Design Decisions”, 5th IEEE/IFIP Working Conference on Software Architecture, pp. 109–119.

[3] Seo C, Edwards G, Malek S, Medvidovic N, (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] Bass L, Clements P, Kazman R, (2003) “Software Architecture in Practice (2nd Edition),” Addison-Wesley.

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

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

[7] Avgeriou P, Zdun U, (2005) “Architectural patterns revisited:a pattern language” ‹Proc. of 10st European Conf. on Pattern Languages of Programs ‹pp.1-39.

[8] Goseva- Popstojanova K, Trivedi K, (2000)” Failure correlation in software reliability models”. IEEE Trans. Rel. 49 1, pp. 37–48.

[9] Kim J S, Garlan D, (2005) "Analyzing Architectural Styles with alloy", Proc. of the ISSTA 2006 workshop on Role of software architecture for testing and analysis, pp. 70-80, 2006.

[10] Reza H, Grant E, () ” Quality-Oriented Software Architecture”, Proc. of the IEEE Conf on Information Technology, Coding and Computing, pp. 140- 145.

[11] Buschmann F, Meunier R, Rohnert H, Sommerlad P, Stal M, (1996) ” Pattern-Oriented Software Architecture- A system of Patterns”. John Wiley & Sons.

[12] Cheung R C, (1980) “A User-Oriented Software Reliability Model”, IEEE Transactions on Software Engineering Vol.6, No 2, pp. 118–125.

[13] Y. Meng-Lai, C.L.Hyde and L.E. James, (2000) ” A Petri-Net Approach For Early-Stage System-Level Software Reliability Estimation”, Proc. of the IEEE Annual Reliability and Maintainability Symposium, , pp. 100-105.

[14] Farr W, (1996) Chapter 3 (Software Reliability Modeling Survey), M.R. Lyu (Editor), Handbook of Software Reliability Engineering, McGraw-Hill, New York.

[15] Silberschatz A, Korth H F, Sudarshan S, (1997) ”Database System Concepts”, McGraw-Hill Series in Computer Science.

[16] Kazman R, Klein M, Barbacci M, Longstaff T, Lipson H, Carriere J, (1998) “ The Architecture Tradeoff Analysis Method”, proc. of the 4th Int. IEEE Conf. on Engineering of Complex Systems. CS Press.,pp.68-78.

[17] IEEE std 610.12-1990 (n.d.) (1990) IEEE Standard Glossary of Software Engineering Terminology.

[18] ISO, “ISO 9126-1:2001,(2001) Software Engineering – Product Quality, Part 1: Quality model”.

[19] Tripathi R, Mall R, (2005) “Early Stage Software Reliability and Design Assessment“, Proc. of the 12th Asia-Pacific Software Engineering Conference (APSEC’05), Dec.

[20] Leblanc S P, Roman P A, (2002) “Reliability Estimation of Hierarchical Software Systems”, Proc. of Annual Reliability and Maintainability Symposium.

[21] Shahmohammadi G.R., (2014) “Evaluation of the Software Architecture Styles From Maintainability Viewpoint”,int. Journal of computer science & information technology, Vol. 6, Issue 1.

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 Amin University (Tehran, Iran). His main research interests are Software Engineering, Software Architecture, Software Metrics, Software Cost Estimation and Software Security.