The Reusability and Coupling Metrics for Service Oriented Softwares

Huynh Quyet Thang, Pham Thi Quynh, Tran Quoc Viet

Abstract- Service-oriented architecture (SOA) has inherited and developed based on some others such as object-oriented and component-based architecture. SOA is a method of developing software which represents reliability, efficiency and maintainability. However, to evaluate these advantages adequately, we need to have metrics to measure quality attributes. Until now, although many scientists have proposed a huge of metrics for object-oriented and component-based software, the evaluation of service-oriented software's quality attributes has not been aware widely. Reuse and couple abilities are the most important attributes of service-oriented software. We assessed and chosen the most suitable metrics with service-oriented software based on comparison between many software architectures. Then, we modified these metrics to apply in SOA. In this paper, we propose two suites of metrics for reuse and couple and a suite of dynamic metrics. Developers with metrics can predict the cost of software development and maintenance. We use Web Service software for testing and evaluating. The results we gained were compared with other metrics applied in service-oriented software. These results applied in Web Service are also right for every SOA.

Keyword- service-oriented software; link, couple, reuse, dynamic metrics;

1 Introduction

Today, SOA is a standard architecture for building modern software and applied widely. SOA is designed for purposes such as integrating easily, use flexibly. In SOA, functions are designed as services which have abilities of reuse and sharing. A service is a part of function, represents its responsibilities via abstract interface, and hides its implementation [13]. A service can be used as a module coupled in a software or combined with others to develop a service-oriented software. Service-oriented software usually performs a fixed business process which divided into some independent sub-processes. These sub-processes are supported by only one or some services. Comparing with software built by older methodologies, service-oriented software has more advantages, for example independence with application foundation, flexibility in couple, and high reusability which help in developing software with higher reliability and cheaper cost.

In software developing approaches, metrics and measurement for evaluating software quality play important roles. Nowadays, these are a few metrics applied in service-oriented software [7, 16].

This work was supported in part by the Vietnam's Ministry of Science and Technology under Grant KHCB2.034.06.

Huynh Quyet Thang is the Head of Software Engineering Department, Hanoi University of Technology, Hanoi, Vietnam (phone: 844-8682595; fax: 844-8692906; e-mail: thanghq@it-hut.edu.vn).

Pham Thi Quynh is with the Dept. of Software Engineering, Faculty of Information Technology, Hanoi National Univerisity of Education, Hanoi, Vietnam (phone: 844-7547324; e-mail: ptquynh@hnue.edu.vn).

Tran Quoc Viet is a engineer of the Oracle Vietnam Pte Ltd (phone: 844-9432595; viettq@gmail.com)

In this paper, we propose suites of metrics which assess service-oriented software according to two aspects: reuse and couple, include static and dynamic metrics. While static metrics evaluate complexity and couple of service-oriented software and each service's reusability, dynamic metrics evaluate software at run time. These metrics can be used in different phases of software process, especially design, test and maintain phases. In addition, project mangers can use them for identifying risk and predict software developing and maintaining cost.

The rest of the paper is organized as follows. Section 2 gives base knowledge, ideas and suites of proposed metrics. Section 3 represents experience and evaluation by comparing with metrics applied in Web-service software and correlative metrics. Section 4 concludes the paper.

2 The suite of metrics and measurement for service-oriented software

Service-oriented software is more preferable than others because of services' flexible couple and high reuse. These are the most important attributes of this software. Therefore, we focus on developing metrics which evaluate two quality attributes: couple metric and reuse metric. We inherit metrics applied in object-oriented and component-based software and modify them to appropriate for service-oriented software. Besides, we also built some new metrics which evaluate its specific characteristics. Next, we give a detail of three metrics: couple metric, reuse metric and dynamic metric.

2.1 Metrics for coupling

In service-oriented software, modules (or services) spread over the Internet or WAN. Service-oriented software will integrate these modules to enable them to link easily. The links in service-oriented software are very flexible and loose, means that they will be created only when linked-services are used which do not occur in every time. The links play an important role and it is said that they are a spine of service-oriented software. Therefore, we need to built a suite of metrics to evaluate the couple between services, in which we focus on the link and the complexity of links of service-oriented software.

a) Link metric

The link is a important property of software, especially in service-oriented software. It is represented via relationships between services in service-oriented software. The more relationships with others a service has, the more contracts with others it has. In this situation, a little change occur in the service which influences on other services in software. The relationships in service-oriented software are expressed by the links. In

summary, a service which has more links with other services will couple with others more highly. We can apply in global relationship in service-oriented software, a software which has more links will has couple more highly.

From this idea, we propose Average Number of Link (ANL) metric.

$$ANL = \frac{\#links}{\#services} \quad (1)$$

where #links is a number of links or relationships between the measured service with other services in software

#services is a number of services included in software or the number of responsibility of measured service.

If the value of ANL is large, the couple of software will be high; and conversely.

b) The suite of complexity of links metric

The complexity is divided into many kinds: size complexity, structure complexity, runtime complexity … In ther paper, we evaluate the complexity of links base on their structure. In SOA, links are connections between services and themselves which are used by methods to transfer data. The data are input arguments and return outputs from methods. We suppose that the evaluation of software's complexity can perform via the complexity of links. The complexity of link is represented by the number of methods which are used in this link and the number of messages which are sent over this link. From this view, to evaluate the complexity of each link, we develop two metrics:

Number of Method per Link – NML:

$$NML(l) = \#methods \quad (2)$$

Number of Parameter per Link – NPL:

$$NPL(l) = \#parameters \quad (3)$$

To evaluate of entire software base on links, we also propose two metrics:

Average of Number of Method per Link – ANML:

$$ANML = \frac{\sum NML(l)}{\#links} \quad (4)$$

Average of Number of Parameter per Link – ANPL:

$$ANPL = \frac{\sum NPL(l)}{\#links} \quad (5)$$

The larger ANML and ANPL 's values are, the larger the complexity of links are. Therefore, software's complexity also is high.

2.2 Reuse metric

SOA has been developed with a hope that will improve the reusability of service. Due to the ability of independence with infrastructure, the service‘s reusability is higher than other modules in software such as class, component … To compare the reusability between services, we need to build specific metrics.

We develop a suite of reuse metrics for service based on reuse metrics for component which were proposed by Washizaki [1]. In 2003, Washiaki introduced a suite of reuse metrics for component which include:

- Existence of meta-information (EMI)
- Rate of Component Observability (RCO)
- Rate of Component Customizability (RCC)
- Self-Completeness of Component’s Return Value (SCCr)
- Self-Completeness of Component’s Parameter (SCCp)

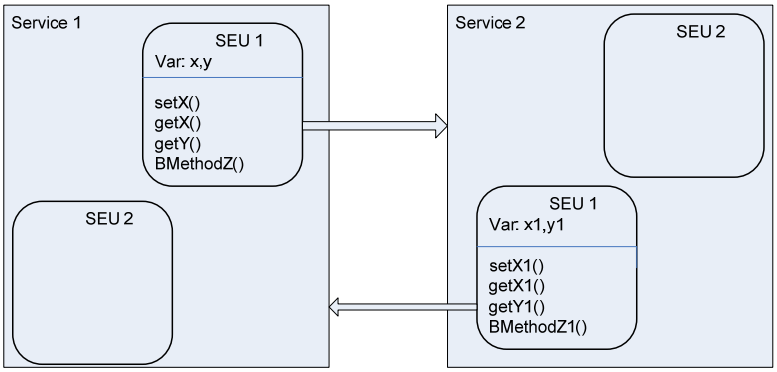

Fig. 1 SOA according to SEU

From Washiaki’s idea, we develop a model of reuse metric for service. This model includes some quality factors in ISO 9126 [9] such as Understandability, Adaptability and Portability. We choose these factors base on the analysis of tasks in black box reused services. Besides, we also consider the Flexibility of service [10]. In conclude, the model of service’s reusability shows later tasks:

- Understand all function of service to decide whether it is necessary to add a new function or not. Users need to understand to perform this task. The understandability is defined by the evaluation about users’ effort which was given to know concepts behind a service and the ability of this service.
- Change a service to identify the functional requirements of a new system. A service need to have a high adaptability to perform this task. The adaptability is a ability of service can change according to a new requirement which is very different from original requirement.
- Flexible with every requirements of users. In a specific situation, users may have some requirements which differ from service’s functions. For instance, users need only the information about customer’s address without other information such as salary, degree … This is very important in data security and system’s

performance. Therefore, instead of providing only response way, service will provide more methods which have the same tasks but they differ about inputs and outputs. It leads to response more contexts according to user's requirement and improve the reusability. The situation is similar to concept of polymorphism in object-oriented programming.

- Move a service into a new environment: Service need to have portability to be performed this task. Portability is an ability which software can be adapted in many different environments. To move into the new environment easily, service need to have the independence with external factors itself.

To inherit Washiaki's metrics, we use the homogeneity between component and service. We can see that a service is created by basic units, such as classes, components … They are considered as service encapsulated units. Each unit is similar to a component. In this view, SOA can be represented via figure 1 (see Figure 1).

Based on Washizaki's metrics and reuse model which was proposed, we build six metrics for reusability of service:

- Exist of Meta Information (EMI): This is specification which describes functions, protocol, interface of service. Value of metric is zero if no specification and one if specification is provided.
- Rate of Service Observability (RSO): Service is created by encapsulated units. Each unit has readable variables. Therefore, percentage of service observability is average of encapsulated units' observability.

$$RSO(S) = \frac{RCO(SEU_1) + ... + RCO(SEU_n)}{n} \quad \textbf{(6)}$$

where RCO(SEUi) is calculated according to RCO for components.

- Rate of Service Customizability (RSC): Similarly, RSC is a percentage of encapsulated units' RCC in a service.

$$RSC(S) = \frac{RCC(SEU_1) + ... + RCC(SEU_n)}{n} \quad \textbf{(7)}$$

where RCC(SEUi) is calculated according to RCC for components.

- Self – Completed of Service's return Value (SCSr): if a business method has not return value, it will not relate with objects which use it. So, this method is more independent and has higher portability.

$$SCSr(S) = \frac{\sum Mbr}{\sum Mb} \quad \textbf{(8)}$$

where Mbr is sum of methods which have not return value and Mb is the sum of method in a encapsulated unit.

- Self – Completed of Service's parameter (SCSp): if a business method has not argument, it will not depend in objects which use it.

$$SCSp(S) = \frac{\sum Mbp}{\sum Mb} \quad (9)$$

where Mbp is sum of methods which have not argument and Mb is the sum of method in a encapsulated unit.

- Density of Multi-Grained Method (DMG): is percentage of the sum of overloaded functions in the sum of different tasks.

$$DMG = \frac{\sum_{i=1}^{n} MG_i}{n} \quad (10)$$

where MGi is the sum of overloaded functions and n is the number of different tasks.

All of metrics have value in range from 0 to 1. If the value of metric is nearly one, the measured service will have high reusability; and reverse.

2.3 Dynamic metrics

Dynamic metrics evaluate software in runtime. These metrics help assess software quality completely. Since 70s, scientists proposed dynamic metrics for software. McCabe [6] built a dynamic metric called Cyclomatic Complexity based on finding all of basic test path. His purpose was finding all of path in flow graph which described model of algorithm. He focused on control and case statements in code to build flow graph of program. Cyclomatic Complexity metric evaluated software's complexity and testability.

In 2005, Narasimhan [2] also proposed dynamic metrics for component - based software. His metrics evaluated components' positive in software where these components were included.

a) The idea of building dynamic metrics for service-oriented software

Based on the evaluation of software's complexity by flow graph, we develop a suite of metrics to assess a business progress. However, compare to McCabe's metric, the basic difference is we evaluate the complexity of business progress instead of the complexity of software's code. The service – oriented software's progress is described in BPEL file included while and case statements. So, this progress will be modeled in a workflow graph, in which each node represents a task or a group of task and each edge represents a workflow. The area is enclosed by edges will be calculated to identify the complexity of workflow.

We build a metric called Service Cyclomatic Complexity (SCC) which is calculated by the number of circles in workflow graph according to formula SCC = E – N + 2 where E is the number of edges and N is the number of nodes in graph. The larger

SCC's value is, the more complex business progress is and reversely. In addition, SCC metric also influence on testability of service-oriented software, because the more circles, the more basic paths which are tested and the more difficult test progress is.

b) Developing a suite of dynamic metrics for service – oriented software

Each metric evaluates only one aspect of quality property. To reflect the quality of software completely, we develop a general metric from incoherent metrics.

The ANL metric shows the couple of software, while we know that the link ability also influences on the complexity. Similarly, the ANML and ANPL metrics also are

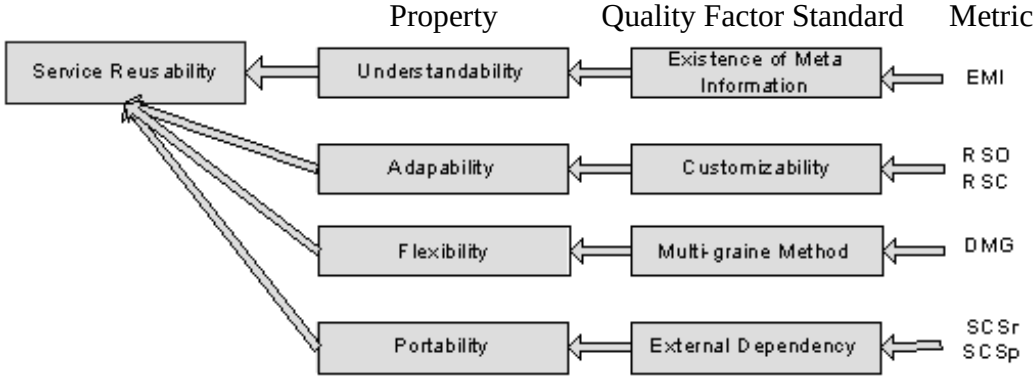

Fig. 2 The relationship between metrics and quality factor

factors which affect on the complexity. Therefore, we propose the general metric to evaluate the complexity of link from ANL, ANML and ANPL metrics.

Link Complexity metric:

$$LC = a.ANL + b.ANML + c.ANPL \quad (11)$$

where a, b, c are adjusted coefficients which suggest the level of impact of three metrics. The value of coefficient are higher, it means that the correlative metric is more important.

To synthesize reuse metrics, we based on the model of the relationship showed in figure 2. According to this model, service's reusability is created from four properties such as understandability, portability, flexibility and adaptability. Each property has particular metrics to assess directly. For example, EMI metric can evaluate the understandability, DMG metric can evaluate the flexibility.

From this point, we propose a general reusability metric for service.

Service Reusability Metric:

$$SRM = a.EMI + b.RSC + c.SCSr + d.DMG \quad (12)$$

where a,b,c,d are adjusted coefficients which can be changed depending on the role of each metric.

3 Experimentation and evaluation

SOA is a theory model and it need to a practical technology which adapts this model. Until now, Web Service has been the most complete and suitable technology. Web Service has been developed by Oracle, IBM, Microsoft. To evaluate metrics practically, we experience metrics proposed in section 2 via Web Service technology.

In Web Service technology, services are described by WSDL files and service-oriented software are showed by BPEL files. There are interfaces which provide functions of service and software. Due to security, service – oriented software is attached with these files instead of detailed design. Therefore, we only calculate some metrics such as ANL, ANML, ANPL, EMI, SCSr, SCSp and SCC as we proposed because these attachment files are not detailed.

To test our metrics whether reflect quality properties completely, we compare our results we gained with the results which were mentioned in papers [7] and [16]. In paper [7], Taixi Xu, Kai Qian and Xihe evaluated de-coupelability of service – based software. This paper gave four metrics DSD, DPD, ARSD and ASIC, in which we used ASIC metric to compare. In paper [17], Mikhail Perepletchikov, Caspar Ryan, and Keith Frampton proposed a suite of CK-extended metrics and LOC metric. We used CBO, NWC, RFC and LOC to compare with our results.

Metrics are classified according to ability of assessment of each quality property: link metrics (ANL, CBO and RFC), complexity metric (ANML, SCC and WML), size metric (ANPL and LOC). We arrange ANPL metric into size metrics because messages are supposed as means move in a path and the density of these means is high which leads to the width of path is large. For reuse metrics, we can not find correlative metrics to compare and have not reused service figures in the reality because of company's security policies; so we will evaluate this suite of metrics via other way. All of metrics were experienced over samples of Oracle and IBM.

The below figures show graphs which compare the results from other metrics. In three figures, horizontal axis presents measured services that are samples of Oracle and IBM and vertical axis presents the value of metric of each services. Figure 3 represents the results from link metrics (ANL and CBO). The results we gained form both of metrics are relatively similar.

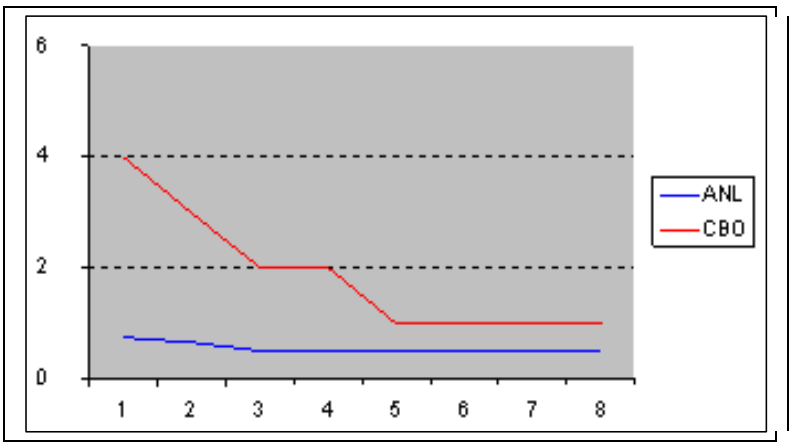

Fig. 3 The comparison between link metrics - ANL and CBO

From figure 4, we can see that the shapes of three graphs are the same except some points. Because each metric reflects each aspect of complexity of software.

Figure 5 shows the graph of reuse metrics include EMI, SCSr and SCSp. The value of SCSp metric always nearly zero which demonstrate that SCSp metric is not suitable with Web Service technology. The value of EMI metric is one because almost of service-oriented software are attached with specification. Based on statistics, we calculate the range of SCSr metric's value from 0.4046 to 0.5046. The value of this range depends on the accuracy and the amount of data which are used to test.

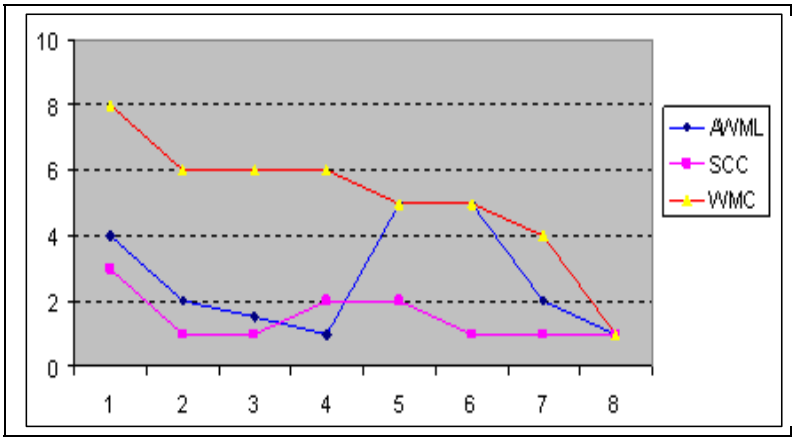

Fig. 4 The comparison between complexity metrics

After experiences, the results gained from our metrics are relatively similar and so even more detailed than pre-proposed metrics.

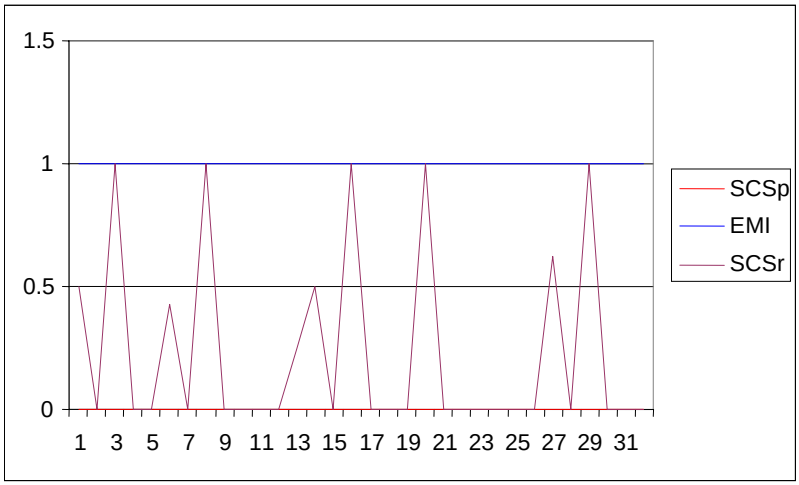

Fig. 5 The graph of reuse metrics

4 Conclusion

In this paper, we have developed three suites of metrics: link, reuse and dynamic metrics. Each suite of metrics reflects a quality property of service – oriented software: link metrics give the complexity of software, reuse metrics show the reusability of software, and dynamic metrics reflect the complexity of business progress.

The experiments also proved the relationship between metrics and relative quality properties. The suite of reuse metrics assess the reusability in which EMI metric evaluates the understandability, RSO and RSC metrics show service's Observability and Customizability, SCSr metric represents the Self-Completeness of service. However, from the experiments, SCSp metric is not suitable with Web Service technology and so we reject it from the suite of reuse metrics. The suite of link metrics contain ANL, AWML and AWPL metrics which measure the linkability, the complexity and size of service – oriented software relatively. The suite of dynamic metrics includes SCC and AS metrics where SCC metric evaluates the software's complexity and AS metric shows the performance of services in software.

We also built a general metric from relative metrics. This general metric not only represents the supporting relation between metrics but also provide a complete model of metrics for users. To make the relationship between metrics and quality properties clearer, we will describe it according to the model metrics – criteria – quality factors. Our metrics have links to criteria which continue to link to quality factors as described in McCabe's model. This model shows the nature of metrics and the level of influence of each metric on quality factors.

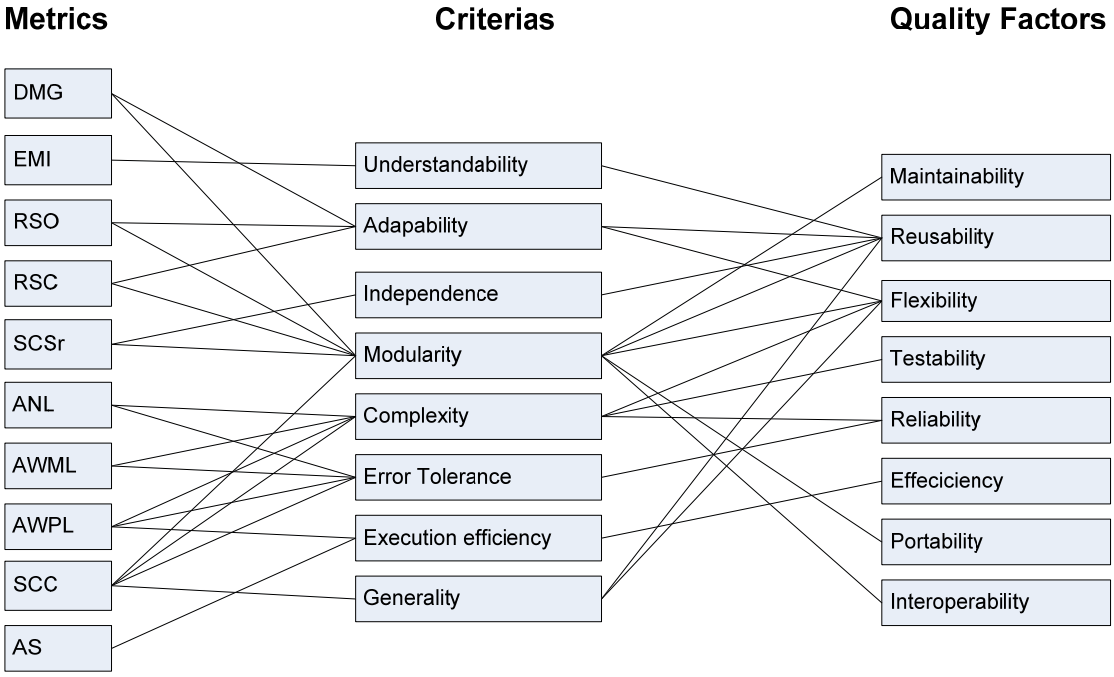

Fig.6 The relationship between Metrics – Criterias – Quality Factors

References

1. Hironori Washizaki, Hirokazu Yamamoto and Yoshiaki Fukazawa: Software Component Metrics and It's Experimental Evaluation. Symposium Empirical Software Eng. (ISESE '02) pp 19-20, 2002.
2 V. Lakshmi Narasimhan, and Bayu Hendradjaya: Theoretical Considerations for Software Component Metrics. Information Sciences, International Journal, Volume 177 , Issue 3 (February 2007), pp 844-864, 2007.
3 Chidamber S. and Kemerer C: A Metrics Suite for Object Oriented Design, ,IEEE Transaction on Software Engineering. Vol.20,No. 6,pp 476 – 493, 1994
4 Marcela Genero, mario Piattini, Coral Calero, ALARCOS Research Group,Universityof Castilla-La mancha, Spain: A Servey of Metrics for UML Class Diagrams. Journal of Object Technology 4:pp 59-92 (2005).
5 Lionel C. Briand, Sandro Morasca, Member, IEEE Computer Society, and Victor R. Basili, Fellow, IEEE: Property - Based Software Engineering Measurement. IEEE Transactions on Software Engineering, Volume 22 , Issue 1, pp 68 - 86 ,1996.
6 Roger S. Pressman, Ph.D: Software Engineering, A Practitioner's Approach. Mc Graw-Hill, 2001, fifth Edition
7 TaiXu, Kai Qian, Xi He: Service Oriented Dynamic Decoupling Metrics. Computer and Information Science, 2006. ICIS-COMSAR 2006. 5th IEEE/ACIS International Conference on , pp 44-47, 2006.
8 Guttorm Sindre and Reidar Conradi , Even-Andr´e Karlsson: The REBOOT Approach to Software Reuse. Journal of Systems and Software, Volume 30 , Issue 3 (September 1995), pp 201 – 212, 1995.
9 Int'l Organization for Standardization: Software Product Evaluation-Quality Characteristics and Guidelines for Their Use. ISO/IEC Standard, ISO-9126, Geneva, 1991.
10 James McGovern, Sameer Tyagi, Michael Stevens and Sunil Matthews: Java Web Services Architecture. Morgan Kaufmann Publishers, 2003.
11 Cern Kaner, Senior Member, IEEE, and Walter P.Bond: Software Engineering Metrics - What do they measure and How do they know. Proceedings of 10th International Software Metrics Symposium. Metrics 2004.
12 Mark Endrei & others: Patterns: Service-Oriented Architecture and Web Services. IBM Press, IBM RedBook, 2004.
13 Micheal S. Mimoso: A defining moment for SOA. http://searchwebservices.techtarget.com/originalContent/0,289142,sid26_gci1017004,00.html, 2004.
14 Schneidewind, N. , IEEE std. 1061-1998: Standard for a Software Quality Metrics Methodology, revision And Reaffirmation. Software Engineering Standards Symposium and Forum, 1997.Emerging International Standardsapos;. ISESS 97., Third IEEE International.
15 Li W. and Henry S: Maintainance Metrics for the Object-Oriented Paradigm. 1st International Software Metrics Symposium,pp. 52-60,1993
16 M. Perepletchikov, C. Ryan, and K. Frampton (RMIT University): Comparing the Impact of Service-Oriented and Object-Oriented Paradigms on the Structural Properties of Software. Proc. of 2nd International Workshop on Modeling Inter-Organizational Systems (MIOS), in conjunction with the OTM 2005, October 2005
17 Connecticut Object – Oriented Users Group: Service – Oriented Architecture. June 10, 2003. Website: http://www.cooug.org/java/.
18 Huynh Quyet Thang, Pham Thi Quynh: Metrics for the Intergration and Reusability of Software Components. AUN/SEED-NET ICT-Field Wise Seminar on Control and Computer. Yogyaakarta - Indonesia, 7-8 August, 2006, pp B89-B99.