SIZE METRICS FOR SERVICE-ORIENTED ARCHITECTURE

Samson Wanjala Munialo[1], Geoffrey Muchiri Muketha[2] and Kelvin Kabeti Omieno[3]

[1]Department of Information Technology, Meru University of Science and technology, Kenya
[2]Department of Information Technology, Murang'a University, Kenya
[3]Department of Information Technology and Informatics, Kaimosi Friends University College

ABSTRACT

Determining the size, effort and cost of Service-oriented Architecture (SOA) systems is important to facilitate project planning and eventually successful implementation of a software project. SOA approach is one of the recent software architectures that allow integration of applications within and outside the organization regardless of heterogeneous technology over a distributed network. A number of research studies have been done to measure SOA size.However, these studies are not based on software metrics rendering them to be ineffective in their estimation. This study defined a set of SOA size metrics based on Unified Modelling Language (UML). The study employed Briand's theoretical validation to test the validity of the designed SOA size metrics. Findings from this study will provide metrics to measure SOA size more accurately and form a basis for future software engineering researchers to develop more effective and more accurate size metrics and effort estimation methods.

KEYWORDS

Service-oriented Architecture, Web services, software metrics, Unified Modelling Language, effort estimation.

1. INTRODUCTION

The ability to measure size and estimate software effort precisely contributes to better management of IT projects and more so software systems. One of the key indicators to be considered when estimating software development effort and cost is software project size. The size of the software project determines the scope and is modelled as the main input when estimating development effort and cost. Software size metrics are standards of measure of the degree to which a software system possesses size-based properties [1]. The goal of using size metrics in software engineering is to obtain objective and quantifiable measurements which may have valuable applications in estimating software development effort. Software developers and Software project managers must tell how big a software project is before estimating software development effort and cost.

Software developers have had the interest of measuring accurately the size of software products. Earlier metrics were based on software lines of codes or function points to estimate the size [2]. However, demand for new functionalities and inclusion of new features such as software re-use, distributed systems and iterative development has established a need for new software size metrics.

Service-oriented Architecture (SOA) is an example of a popular paradigm for developing distributed systems that provides a challenge to existing software size metrics. SOA consists of service providers which are elements that offer services to be used by other service users. The need for agility, cost-effectiveness and efficiency, adaptability and legacy leverage in the rapidly changing business environment has led many organizations to migrate to SOA applications. SOA other benefit include clear separation of services from implementation which allows service upgrades to occur without overhaul of the entire system and less impact on service users [3]. SOA is an important aspect of organization's IT infrastructure as it links all applications and services that support business processes. SOA is a paradigm shift in designing information systems where services corresponding to business functions are published in form of standard interface to be discovered by other services [4].

Size measurement has been identified as one of the key attributes in software estimation [5]. Measuring SOA systems size is difficult because SOA comprise of integration among services within and outside the organization regardless of heterogeneous technology and programming language over a distributed network. Some of the most frequently used software size metrics in use since 1980's includes Source Line of Code (SLOC) and Function Point [6]. These metrics are challenged when dealing with SOA characteristics such as loose coupling, SOA internal structure, service composition and messaging. This prompted the introduction of SOA metrics such as number of service count, and service interface count [6] [7] [8] [9][10] which are complexity metrics rather than size metrics and they are not weighted appropriately with regards to SOA size. In addition, COSMIC- SOA size metric was introduced to measure SOA size. This metric howeverrelies on counting the amount of data movements among services to measure SOA system size disregarding other size based attributes. Therefore, there is a need to introduce metrics that will take into consideration multiple size attributes that can be measuredin SOA.

This paper seeks to define a suite of size metrics that will be used to measure the size attributes of SOA software based on UML interface and sequence diagrams. It provides an analysis of existing software size metrics and use a case example to illustrate the defined metrics.Lastly, theoretical validation of the proposed metrics isbased on Briand's size properties framework.

2. RELATED WORK

Over the decades, software development processeshave transformed from structured designs such as the waterfall to new approaches such as agile, component-based, software re-use and service-oriented architecture. This transformation resulted to increase in software size, and complexity. Software effort factors have also changed over time due to evolution in software practices [8]. Software size has been the main software effort indicators since early 1980's with the introduction of Source Lines of Codes (SLOC) and Function Point analysis metrics to measure software size. Later on, Modular programming paradigm such as object-oriented programming shifted researcher focus from lines of codes to internal structures and relationships among modules as the main attributes when measuring software size. Existing SOA metrics such as number of service count, service interface count [9] andfunctional size measurement method for SOA[12] were built on the foundation of earlier programming architectures.

SLOC is a parametric used to measure the size of a program by counting the number of lines of a program's code [10]. The goal it to measure the amount of intellectual work put into program development. SLOC is dependent on programming language platform, and therefore, it is inadequate when dealing with heterogeneous systems developed using different programming platforms such as SOA. Software effort estimation methods that use SLOC as size metrics include the four versions of COCOMO models.

Function Point metric measures the number of functionality in a software application [10]. Function Point metric is independent on programming language and it is arrived at by counting the number of five basic software components including external inputs, external outputs, logical internal files,external interfaces and external inquiries. Each of the function is weighed by complexity factor ranging from low, average to high [10]. Each function component is multiplied with a respective complexity level then summed up to give Function Count (FC). Function Point can be applied at requirement specification or design phase of system development life cycle.Function point works best when requirements are well defined and there is certainty on the structure of system to be developed.

SeveralFunction Points have been introduced since 1985 including Mark II Function Points, 3D Function Points, COSMIC full Function Points, De-Marco Function Points and Feature Function Points [11] and has been reviewed further by the International Function Point User Group (IFPUG). However, most Function points versions do not capture SOA features such as loose coupling, distributed systems and how to estimate different types of services. This prompted adjustment of traditional Function Points to take SOA features into consideration [12]. They introduced a Function Point version calibrated to meet SOA features by adjusting data communications, distributed data processing, performance, and heavily used configuration to match SOA characteristics. Based on case studies involving three projects the proposed function point method returned more accurate results as compared to the traditional function point method[12]. However, service type, service structure and dependency were not captured and detailed analysis of the method was not documented.

Object-Oriented programming shares a number of properties with SOA and component-based systems including the separation of tasks into methods or operations, cohesion and other properties. Similarities between object-oriented and SOA properties have prompted a number of SOA metric researchers to adopt object-oriented metrics. Traditional object-oriented metrics were used to measure SOA complexity attributes [6]including the Weighted Methods Count (WMC) [13] which counts the number of weighted methodsbased on McCabe’s cyclomatic complexity [14]. Other metrics adopted by SOA researchers from Object-oriented are coupling and cohesion metrics. Furthermore, Object-oriented designs tool such as UML is widely used to represent SOA design. A case in point is Service-oriented Architecture Modeling Language (SoaML) [15] which is an extension of UML.

Existing SOA complexity and size metrics include Weighted Service Interface Count (WSIC), number of services metrics and COSMIC-SOA metrics. WSIC was proposed to measure the number of exposed interfaces or operations as defined in the Web service description language (WSDL) documents [7]. WSIC returns the number of operations in a service based on the hypothesis that the higher the number of service operations the more complex a SOA will be. WSIC provided an insight on the relevance of operations as an attribute in determining SOA complexity. However, empirical analysis was not done to validate the metric, it focuses on SOA complexity rather than size and other attributes such as operation parameters are not captured in the metric. Number of services (NS) metric is a simple count of services contained in a SOA system [6][7] [9]. No empirical validation study was associated with this metric in these studies. The major limitation with this metric is that it simply countsthe number of services and disregards the fact that these services could be having different complexity and features.

COSMIC-FFP (Common Software Measurement International Consortium-Full function Points) was introduced to measure functional size of software. COSMIC measurement metrics involves applying a set of models, principles, processes and rules to measure functional user requirements of a given software which will result to the function size of software [18]. The group has published several metrics which include COSMIC-FFP for web applications and COSMIC-SOA for SOA based projects.

COSMIC principle states that the main programming efforts are dedicated towards handling of data movements from/to the storage and users [19]. COSMIC metrics require a definition of the context model for a specific application with clear boundaries separating the software from its operating environment. Each data movement crossing the boundary is counted to give the full function points. The data movements are classified as Entry, Exit, Read and write. One advantage of COSMIC is the ability to estimate software size of big projects by counting the amount of data. However, COSMIC methods focus more on data movement rather than considering SOA size attributes such as operations, dependency and number of services.

3. SOA SIZE METRICS

This study considers SOA internal structure, data movement, interaction and relationship among services as key attributes for defining SOA size metrics[20][21]. The level of abstraction is based on UML static (design-time) design and run-time (dynamic) design. Static metrics are taken from design phase level while dynamic metrics are derived when designing the system run-time model [22][23]. SOA being an architectural style that enables development of services that are modular and integrated can be represented at different levels of abstractions using UML diagram and other extensions of UML such as SOAML [15].

This study defines four metrics namely Weighted Operation Count (WOC), Service Dependency Count (SDC), Weighted Service Count (WSC) metric at static level exposed through SOAML or UML class diagrams and Weighted Message Count (WMC) metrics at run-time level exposed through UML sequence diagram. Furthermore, SOA size attributes are classified into two categories namely service level metrics and systems level metric. WOC, SDC and WMC are service level metrics while WSC metric is a systems level metric.

3.1 Weighted Operation Count (WOC)

Weighted Operation Count (WOC) metric evaluates the internal structure of an individual service by counting the number of operations or methods contained in a service based on their complexity. WOC takes into consideration the number of operations, operations' complexity and operations' parameters to determine the size of a service.

Definition

WOC is defined as a set of operations in a service and a set of parameters contained in an operation.

Therefore, $WOC(S) = < O, P > where\ O \in S \wedge P \in O$

S denotes a service, O is a set of operations and P is a set of parameters in an operation. WOC is based on the hypotheses that the more the number of operations and complexity of operations the greater the size of a service. Consequently, it takes more effort to construct a service with more and complex operations as compared to a service with fewer and simple operations. WOC metric takes into account the number of parameters in an operation as an indicator of more processes to be done by the operation. The WOC metric counts the number of operations weighted according to their complexity and the number of parameters in an operation as shown below.

$$WOC(S) = < O, P >= \sum_{i=1}^{n} (O_i + P_i)$$

This study adopted Albrecht's Function Point Analysis metric [10] weights of related software attributes which were validated and have been used over time by the industry. Based on Albrecht's Function Point Analysis method [10] Internal Logical files (ILF) attribute, this study assigned weights of 5 to simple operation, 7 to average operation and 10 to complex operation as shown in Table 1. In addition, a weight of 1 is allocated to each parameter contained in an operation.

Table 1: Service Operation weight

Operation type	**Examples**	**Weight**
Simple	Get/ Write operation, Arithmetic Calculations, Simple decision making process	5
Average	Operations based on simple algorithm e.g. Searching, sorting.	7
Complex	Operations based on intelligence techniques, decision support algorithm.	10

WOC is a service level metric captured at static level design based on SOAML service interface diagram which reveals operations and parameters graphically. A sample of SOAML service interface diagram showing service operations and parameters is as shown below in Figure 1.

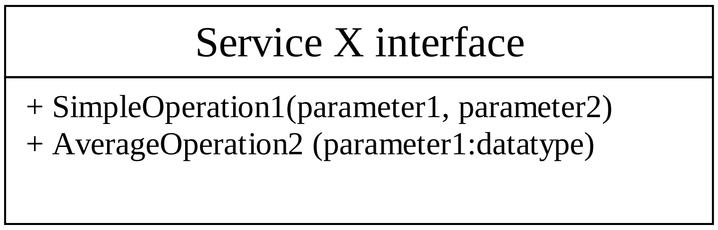

Service X interface
+ SimpleOperation1(parameter1, parameter2)
+ AverageOperation2 (parameter1:datatype)

Figure 1: Service interface diagram

The proposed Weighted Operations Count (WOC) considers a service interface X in figure 1, containing two operations namely +simpleOperation1() with 2 parameters and +averageOperation2() with one parameter.

$$WOC(S) = < O, P >= \sum_{i=1}^{n} (O_i + P_i)$$

WOC(X) = simpleOperation1() + averageOperation2() web service points

simpleOperation1() = O + P = 5 + 2 = 7 web service points

averageOperation2() = O + P = 7 + 1 = 8 web service points

WOC (X) = 7+8 = 15 Web service points

The size of service X based on its internal structure as revealed by SOAML service interface diagram is 15 web service points according to WOC metrics.

3.2 Service Dependency Count (SDC)

Service dependency also known as coupling is the degree of interaction and extent of dependency between services [16]. This study identified Service Dependency attribute as an indicator of web service size measurement. A service-oriented principle states that services should be "loosely coupled" meaning the nature of interaction should be limited to solely exposing operations for interaction with other services. The study defined Service Dependency Count (SDC) Metrics to count the number weighted dependencies in relation to the size of a service based on WOC.

SDC captures dependency attributes from UML static design class diagram at service level. SDC focuses on direct dependency which is dependency that exists between a service provider and a service consumer. Based on this study hypothesis, a service with more interaction will have more configurations to link to other services as compared to a service that is linked to fewer services. Implying that a service is bigger in size when it depends on more services or other services depends on it and more effort is spent when configuring a service dependency.

SDC takes into consideration fan-in and fan-out dependency by counting fan-in dependencies as one dependency given that a common configuration in the provider-side will enable connection from various consumer services. Fan-in dependency has a higher weighting of 4 as compared to fan-out dependencies. On the other hand, fan-out considers the number of dependencies and how deep services are related also known as service composition. Service composition is a collection of related services that take part in solving a specific business function [6].

Services that are not in composition are said to be atomic in which case they do not require other services to complete a business process. When the relationship between services in a composition is deep then the type of composition is strong composition or else the composition is lighter aggregation. Fan-out dependency attributes according to UML representation are classified based on service composition as atomic point-to-point message exchange, lighter aggregation and strong composition weighted 1, 2 and 3 respectively as shown in Table 2.

Table 2 Weighted Service Aggregation table

Composition type	**Weight (Points)**
Atomic (point-to-point dependency)	1
Lighter aggregation	2
Strong composition	3

Definition

WDC is defined as a set of types of dependencies among services.

Therefore, $SDC\ (A)\ = < S, D > \ \ where\ S\ \in A\ \Lambda\ D\ \in S$

Where A is a service-oriented Architecture application, S is a service and D represent sets of dependencies.

Given a service X, $D(X)\ = < Ix, Ox >$ and $S(X)\ =\ WOC(X)$.

D(X) is a set of dependencies on a service X, I is the fan-in dependency node and O is a set of fan-out dependency nodes of a service X. Fan-out dependency is classified into atomic, lighter aggregation and strong aggregation.

Therefore, $O \ = < a, g, t >$

a is a set of Atomic dependency, g is a set of lighter aggregation dependency and t is a set of strong aggregation dependency.In reality, codes that provide functions to link to other services takes a fraction of a service size.Therefore,

$$SDC \ = \ S(D/(D+S))$$

Atomic dependency is indicated by a dotted arrow in UML diagram while lighter aggregation and strong composition are denoted by a light diamond arrow and a dark diamond arrow respectively as shown in UML class diagram in Figure 2.

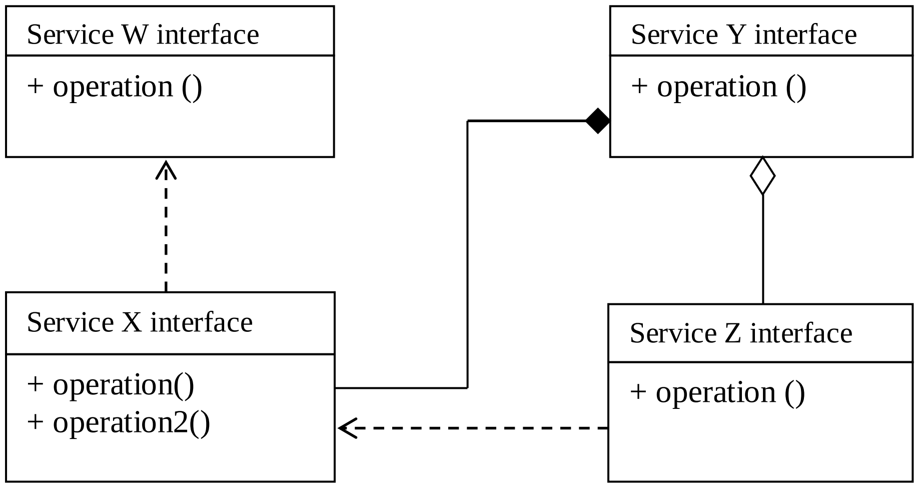


Figure 2: UML Diagram showing dependency among services

Figure 2 represents 4 service interfaces with X service interface depending on W service interface in an atomic dependency interaction as indicated by dotted arrow. Service X depends on service Y in a strong composition relationship denoted by a dark diamond arrow. On the other hand, service Z depends on service Y in a lighter aggregation relationship denoted by a light diamond arrow and depends on X in an atomic dependency relationship. The proposed Service Dependency Count (SDC) considers a service interface S, with dependencies $D_1 \ldots D_n$ identified in the static service interface UML diagram.

Therefore SDC for service X is, $SDC(X) \ = < S, D > \ = \ S(D/(D+S))$

Based on WOC(X) in figure 1, $S(X) \ = \ WOC(X) \ = \ 15 \, web \; service \; points.$

$$D(X) = p + \sum_{i=1}^{n} a + \sum_{i=1}^{n} g + \sum_{i=1}^{n} t$$

$$= \ 4 \ + \ 1 \ + \ 0 \ + \ 3 \ = \ 8 \, web \; service \; points$$

$$SDC(X) \ = \ S(D/(D+S)) = \ 15(8/8 + \ 15)) \ = \ 5.23 \, web \; service \; points$$

Where p is fan-in dependency with a weight of 4, a is atomic fan-out dependency weighted 1 points, g is lighter aggregation fan-out dependencies with a weight of 2 points and t is strong composition fan-out dependencies with a weight of 3 points. Based on SDC, SDC(X) IS 5.23 web service points which is a fraction of WOC(X).

3.3 Weighted Message Count

Weighted Message Count (WMC) represents movement of data groups between services, databases and other applications. Weighted Message is a service level metric that takes into consideration the type of message call classified as synchronous, asynchronous and reply messages. In this regard, data movement specification is linked to the design of information model which is represented by UML sequence diagram. Based on this study hypothesis, there is a relationship between the number of messages in an application and the size of the SOA application because is it takes a process to produce a message.

Furthermore, more weight is assigned to synchronous message call because it requires coordination of events to enable message movements in unison. On the other hand, asynchronous message call is assigned lesser weight due its simplicity in design, it does not return a value and no coordination is required to facilitate data movement as compared to synchronous message call. Lastly, reply message carries much lesser weight based on the fact that reply messages are based on conditional tests that will provide error messages or acceptance messages. Weighted Message Count (WMC) weights and arrows are illustrated inTable 3.

Table 3 Weighted types of process

Message type	**UML arrow line**	**Weight**
Synchronous	———▶	2
Asynchronous	———>	1
Reply message	– – – – ▶	0.5

Definition

$$WMC\ (S)\ = < M >\ where\ M \in S$$

In this case, M represents a set of messages. M is made up of three types of messages that is synchronous, asynchronous and reply message. Based on UML sequence diagram, data groups to and from a service are identified as shown in Figure 3.

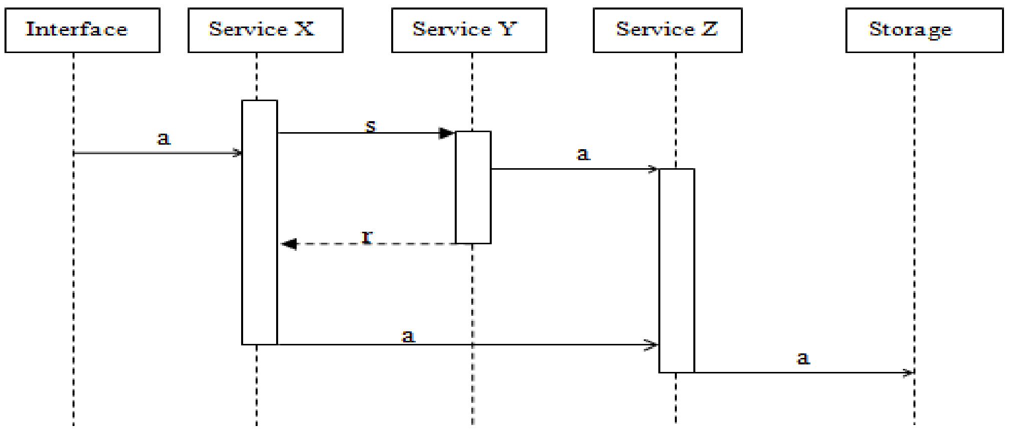


Figure 3: UML sequence diagram showing data movement

From the sequence diagram in Figure 3, WMC deals with message out disregarding incoming messages based on the principle that it takes a process to produce a message. In this regard, service X receives asynchronous massage a from the interface and reply message r from service Y then processes the received messages to give out message $a$and s. Therefore a and$s$ will be counted as shown below.

$$WMC(X) = \sum_{i=1}^{n} s + \sum_{i=1}^{n} a + \sum_{i=1}^{n} r$$

$$WMC\ (X)\ =\ s\ +\ a\ +\ r\ =\ 2 + 1\ +\ 0\ =\ 3\ Web\ service\ points$$

a = asynchronous message s = synchronous messages r = reply message
Given that asynchronous has a weight of 1, synchronous has a weight of 2 and reply message has a weight of 0.5. The total WMC for service X is 3 web service points. Note that reply and asynchronous incoming messages are not counted.

3.4 Weighted Service Count (WSC)

Weighted Service Count (WSC) metric simply sums output derived from WOC, SDC and WMC.

$$WSC\ =<\ WOC, SDC, WMC >$$

Where,

- WOC is a set of all weighted operations and parameters contained in the operations.
- SDC is a set of all dependencies between a service and other services weighted with regard to the type of composition.
- WMC is a set of all messages to and from a service weighted according to the type of message.

WSC is a systems level metric whose attributes are derived from UML static design diagram. WSC returns size of a service system and eventually the size of SOA application measured in web service points.

Definition

$WSC\ =<\ WOC, SDC, WMC >$

According to WSC hypothesis, the more the number of weighted services, dependencies and messages contained in a SOA application, the bigger and more complex the SOA application will be resulting to more effort required to build and integrate services.

Given service X in UML diagram in figure 1, 2 and 3,

$$WSC\ (X)\ =\ WOC(X) +\ WDC(X)\ +\ WMC(X)$$

$$WSC(X) =\ 15 + 5.2\ +\ 3\ =\ \mathbf{23.2}\ \boldsymbol{web\ service\ points}$$

The same concept is applied to other service to calculate WSC for each service then summed to give the total size of SOA application. A case in which the proposed SOA size metrics is applied to a purchase order SOA system is discussed below.

4. Purchase Order SOA System: A Case Study

Based on SOAML design methodology, design of a SOA system starts with business process modelling which involves capturing the business design from an understanding of business requirements and objectives (Amsden, 2010). The business requirements and objectives are translated into business process specification using Business Process Modelling Notation (BPMN). Services are then identified from the business processes and service specifications are captured through UML diagram to identify methods, data movement and relationship among services.

With regard to purchase order process, a consortium of companies needed to align their purchase order processes to business requirements. The Purchase order business processes include managing purchase order, production scheduling, inventory management, shipping and invoicing. The order process starts by receiving and processing the purchase order which includes items description, items quantity and customer details. The purchase order provides information to calculate the price of items, process production schedule and shipping details. Invoice is then prepared by including the total cost of items, production cost and shipping cost.

To measure the size of purchase order process SOA system, Business Process Modelling Notation was converted to UML diagram for a detailed and lower abstraction. UML diagram was used to expose number of services, number of operations, number of parameters, type of relationship among services and data movements among services in a SOA system as shown in Figure 5 and Figure 6. WOC and SDC attributes were captured from UML interface diagram in Figure 5 while WMC attributes were captured from UML sequence diagram in figure 6.

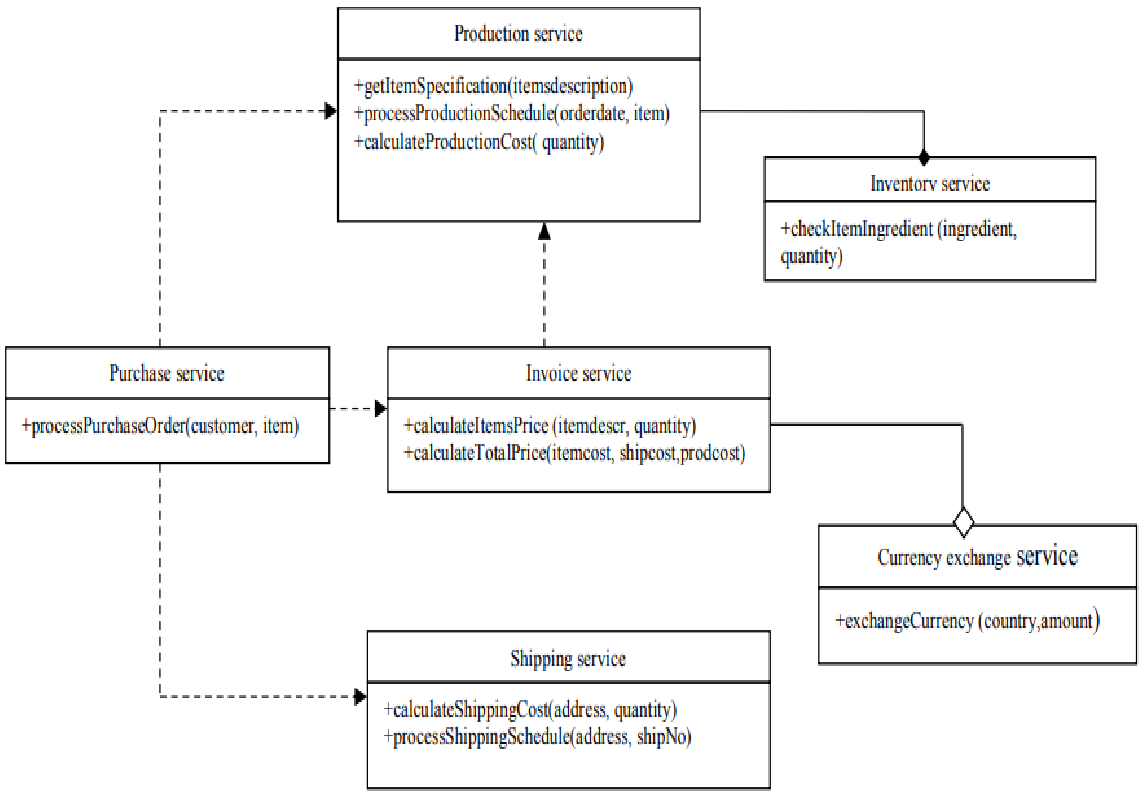

Production service
+getItemSpecification(itemsdescription)
+processProductionSchedule(orderdate, item)
+calculateProductionCost(quantity)
Inventory service
+checkItemIngredient (ingredient, quantity)
Purchase service
+processPurchaseOrder(customer, item)
Invoice service
+calculateItemsPrice (itemdescr, quantity)
+calculateTotalPrice(itemcost, shipcost,prodcost)
Currency exchange service
+exchangeCurrency (country,amount)
Shipping service
+calculateShippingCost(address, quantity)
+processShippingSchedule(address, shipNo)

Figure 5: UML interface diagram representing purchase order process services

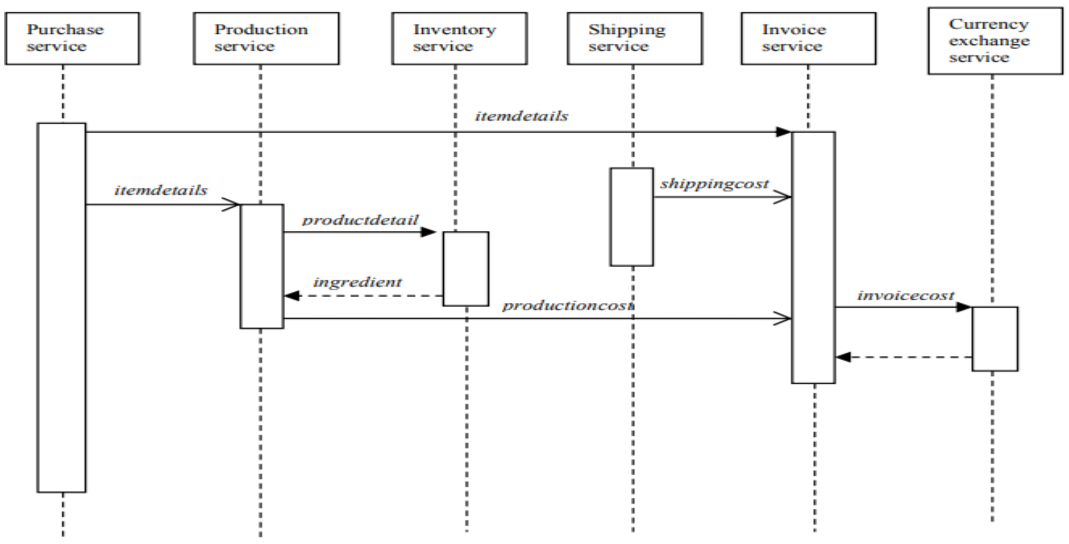


Figure 6 : UML sequence diagram representing purchase order process services

4.1 WOC for purchase order SOA system

Based on SOAML service interface diagram representing the purchase order SOA system in figure 5, WOC considered the number of operations contained in each service, complexity of each operation and the number of parameters as key attributes when determining the size of a service with regard to a service internal structure. For example WOC for invoice service with two operations with simple arithmetic calculation is classified as simple operations each allocated a weight of 5. The first operation has 2 parameters and the second operation has 3 parameters. Therefore,

$$WOC\ (Invoice\ service)\ =\sum_{i=1}^{n}(o_i+p_i)$$

$$=\quad (5\ +\ 2)\ +(\ 5\ +3)\ =\ 15\ web\ service\ points$$

WOC for purchase service, production service, shipping service, inventory service and exchange currency services are as shown in Table 4.

Table 4: WOC for purchase order SOA systems

Service	Operation	Type	Weight per operation	Number of Parameters	Web Service points
Invoice	*calculateItemsPrice*	Simple	5	2	15
	calculateTotalPrice	Simple	5	3	
Purchase	*processPurchaseOrder*	Simple	5	2	7
Production	*getItemSpecification*	Simple	5	1	21
	processProductionSchedule	Average	7	2	
	CalculateProductionCost	Simple	5	1	
Shipping	*calculateShippingCost*	Simple	5	2	16
	processShippingSchedule	Average	7	2	
Inventory	*checkItemIngredient*	Simple	5	2	7
Exchange currency	*exchangeCurrency*	Simple	5	2	7
Total WOC					**73 points**

4.2 SDC For Purchase Order SOA System

Secondly, SDC measured the size of a SOA system by considering the relationship among services. SDC measured the number of fan-out, direct dependencies and type of dependency between services as indicated by different types of arrows linking consumer services with provider services.

For instance, invoice service in figure 5 depends on currency exchange service to perform currency conversion, it depends on shipping service to provide shipping cost and production service to provide production cost. Invoice service has one fan-in dependency and 3 fan-out dependency and the types of fan-out dependency are one lighter aggregate dependency with a weight of 2 and 1 atomic dependency allocated a weight of 1.

Therefore, $SDC\ (Invoice)\ = < S, D >\ =\ S(invoice)\ +\ D(invoice)$

$$S(invoice)\ =\ WOC(invoice)\ =\ 15\ web\ service\ points.$$

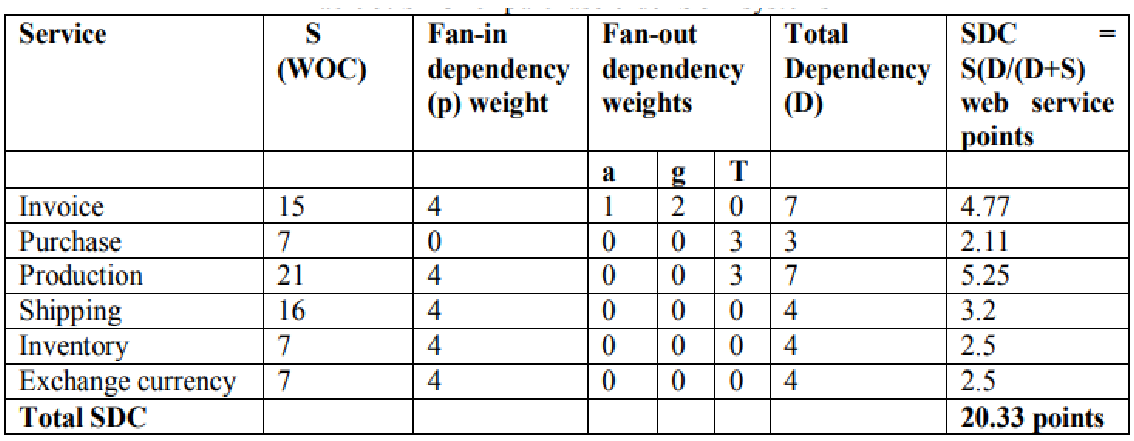
$$D(invoice) = p + \sum_{i=1}^{n} a + \sum_{i=1}^{n} g + \sum_{i=1}^{n} t$$

$$=\ 4\ +\ 2\ +\ 1\ + 0\ =\ 7 \text{ web service points}$$

$$SDC\ (invoice)\ =\ S(D/(D+S))\ =\ 15(7/(7+15))\ =\ 4.77\ web\ service\ points.$$

SDC for purchase service, production service, shipping service, inventory service and exchange currency services are as shown in Table 5.

Table 5: SDC for purchase order SOA systems



Service	S (WOC)	Fan-in dependency (p) weight	Fan-out dependency weights			Total Dependency (D)	SDC = S(D/(D+S) web service points
			a	g	T		
Invoice	15	4	1	2	0	7	4.77
Purchase	7	0	0	0	3	3	2.11
Production	21	4	0	0	3	7	5.25
Shipping	16	4	0	0	0	4	3.2
Inventory	7	4	0	0	0	4	2.5
Exchange currency	7	4	0	0	0	4	2.5
Total SDC							**20.33 points**

4.3 WMC for purchase order SOA system

Thirdly, WMC counted the number of message movements from a service based on the type of message captured by the UML sequence diagram in figure 6 showing different types of arrow lines representing different types of messages. For instance, according to the sequence UML diagram in figure 6, invoice service provides1 one synchronous message, Therefore,

$$WMC(invoice) = \sum_{i=1}^{n} s + \sum_{i=1}^{n} a + \sum_{i=1}^{n} r$$

$$WMC\ (invoice)\ =\ 2\ +\ 0\ +\ 0\ =\ 2\ web\ service\ points$$

Given that asynchronous has a weight of 1, synchronous has a weight of 2 and reply message has a weight of 0.5.

Table 6: WMC for purchase order SOA system

Service	**Synchronous (s)**	**Asynchronous (a)**	**Reply message (r)**	**WMC**
Invoice	2	0	0	2
Purchase	2	1	0	3
Production	2	1	0	3
Shipping	0	1	0	1
Inventory	0	0	0.5	0.5
Exchange currency	0	0	0.5	0.5
Total SDC				**10 points**

4.4 WSC for purchase order SOA system

Lastly WSC will sum the results of WOC, WDC and WMC to give the final SOA size measure for the purchase order SOA system.

$$WSC\ =\ WOC\ \ +\ SDC\ \ +\ WMC\ 73 + 20.33 + 10 = 103.33\ Web\ service\ points$$

The result reflects the size of purchase order SOA system based on attributes revealed by UML interface diagram and sequence diagram.

5. Theoretical Validation

Metrics development involves 2 stages which include metrics definition and metrics validation [23]. Metric definition is the actual design of the metrics through identification of key factors and their contribution to the metric. On the other hand, metrics validation is determining the validity of software metrics with respect to the domain under research. There are two types of software metrics validity namely theoretical validity and empirical validity [24].

Theoretical validation confirms if a metric measures what it is supposed to measure by considering that the new metric does not violate measurement theory. It establishes construct validity by checking whether the new metrics are structurally valid and if the metric respects set criteria as defined by the theory. Most popular theoretical validation frameworks in software engineering include Weyuker's properties and Briand's properties. Weyuker's properties establish validity of software complexity metrics while Briand's properties determine validity of software size and length metrics [23][24]. Empirical validation on the other hand, proves that the measured values are consistent with the values predicted by the new metrics.

In this study, Briand's size properties [25] form the basis of theoretical validation. They proposed a rigorous mathematical framework based on precise mathematical concept with regard to software size, length and complexity measurements [25]. They defined a concept that takes into consideration a system as an entity that has elements and relationships among elements.

According Briand's size properties framework, the size of a system S is a function of size (S) containing sets of elements (E) and sets of relationship among elements (R). The framework defines three fundamental size properties that determine the validity of a software metrics. The three properties include non-negativity, null value and module additivity summarized as follows:

Size Property 1: Non-negativity – The size of a system S is non-negative.

$$S = <E,R> \ where \ S >= 0$$

Size Property 2: Null value – The size of a system (S) is null if elements (E) is empty.

$$S = <E,R> \ where \ S = 0 \ if \ E = 0$$

Size Property 3: Additivity – The size of a system (S) is equal to the sizes of its elements.

$$S = \Sigma(E)$$

5.1 Weighted Operation Count (WOC) Theoretical Validation

Based on Briand's size property framework, a software size metric should satisfy non-negativity, null value and additivity properties to confirm a metric's theoretical validity. In this respect, WOC = < O, P> is non-negative such that the size of service operations and parameters cannot be negative. Given a service S = <O,P> where $O \in S \wedge P \in O$. WOC(S) involves counting the number of operations and parameters which in this case cannot return a negative value. Therefore, WOC(S) ≥ 0 satisfy Briand's size 1 property which states that the size of a system is non-negative.

Secondly, according to Briand's size 2 property, a service must have an operation for its size to count. According to WOC, a service size is determined by the number and complexity of operations and parameters. WOC(S) = <O,P> such that if $O=\emptyset$, then WOC(S) = $\emptyset$ conforming to Briand's size 2 property which states that the size of a system is null if it has empty modules (E).
Thirdly, Briand's size 3 additivity property requires that the size of a system should be equal to the total size of all modules. With WOC case, the size of a service is equivalent to the sum of the size of all weighted operations and parameters contained in a service. The size of a service (S) according to WOC is not greater than the size of all operations contained in a service.

WOC(S) = $M_1+M_2+ \ldots M_n$. Where $M_1=<O_1,P_1>$, $M_2=<O_2,P_2>$ and $M_n=<O_n,P_n>$

Where M represents a set of operations and parameters, O represents weighted operations and P represents parameters. WOC metric meets Briand's size 3 additivity property which demands the sum of a system to be equal to the sum of all modules.

5.2 ServiceDependency Count (SDC) theoretical validation

SDC theoretical validation is based on Briand's property framework to confirm the metric validity. SDC conforms to Briand's size 1 property given that SDC cannot return a negative value as it involves adding weighted operation count and dependencies.

$$WDC\ (A) = <S,D> \ where \ S \in A \wedge D \in S.$$

Therefore, WDC (A) ≥ 0 because $S\geq0$ and $D\geq0$ where A represents SOA application, S represents sets of services and D represents sets of dependencies. Given that the value of S is equivalent to WOC(S), the value of S cannot be negative. Consequently, the value of adding dependencies cannot return negative values.

Secondly, SDC meets Briand's size 2 null value property because when there is no service (S), SDC will return a null value. When S = ∅ then SDC =∅. Thirdly, the value of SDC a is equivalent to the sum of all services and dependencies of services. SDC (A) = $S_1D_1+S_2D_2+$……… S_nD_n conforming to Briand's size 3 additivity property which states that the size of a SOA System (S) is equivalent to the sizes of its elements.

5.3 Weighted Message Count (WMC) theoretical validation

Weighted Message Count (WMC) considers the amount of message exchange among services as indicator of size. WMC counts the number of weighted messages to determine the size of a service. In this regard, If M represents message originating from a service, WMC (M) cannot return a negative value therefore,

$$WMC = < M > \geq 0$$

WMC (M) satisfy Briand's non-negativity property given that the size of a service is non-negative as it results from summing messages from a service which cannot be negative. Secondly, WMC metric returns a null value if there is null message originating from a service. Therefore,

$$If\ M = \varnothing then\ WMC\ (A)\ =\ \varnothing.$$

which conforms to Briand's Null valueproperty which states that the size of a system (S) is null if element (M) is empty. Thirdly, Briand's size 3 property demands that the size of a system should be equal to the sizes of its elements. WMC meets the size 3 property requirements given that, the size of a service S is equal to the sum of the sizes of its messages. S= <M> is equal to the size of S_1 = <M_1>, S_2 = <M_2> …. S_n =<M_n>.

5.4 Weighted Service Count (WSC) theoretical validation

Weighted Service Count provides a framework for summing up results from WOC, SDC and WMC. Based on Briand's size 1 non-negativity property, WSC cannot return a negative value because all the WSC ingredients cannot return a negative value. Secondly WSC = ∅ if WOC Λ SDC Λ WMC = ∅ conforming to the size 2 property. Lastly the size of a SOA application is equivalent to the sum of all services WOC, SDC and WMC conforming to Briand's size 3 property.

WSC (A) = WOC(S_1), SDC(S_1), WMC(S_1) + WOC(S_2), SDC(S_2), WMC(S_2) …… WOC(S_n), SDC(S_n), WMC(S_n)

The size of a SOA application is a result summation of all services WOC, SDC and WMC.

6. Conclusion

Theoretical validation of the SOA size metrics proved the validity of the metrics in measuring SOA size. The metrics derived from UML diagram provided a framework for identifying key attributes relevant for measuring SOA size. The metrics were designed and applied to a purchase systems example to show the metrics applicability. To establish construct validity, the metrics were subjected to Briand's property framework to establish if the metrics are structurally valid. Later on in this study, an empirical validation will be carried out to prove that the metrics results are consistent with the predicted results.

REFERENCES

[1] Coelho, E. &Basu, A., (2012) Effort Estimation in Agile Software Development using Story Points, International Journal of Applied Information Systems

[2] Litoriya, R.,& Kothari, A. (2013) An Efficient Approach for Agile Web Based Project Estimation: AgileMOW, International Journal of Computer Science and Computer applications.

[3] Farrag, A.E. &Moawad, R. (2014). Phased Effort Estimation of legacy Systems Migration to SOA,International Journal of Computer and Information Technology.Ahmed, N.A., & Ahmed A.H. (2012). Enabling Complexity Use Case Function point on SOA. 2013 International conference on Computing, Electrical and Electronic Engineering.

[4] Seth, A., Agarwal, H., & Singla, A. (2010) Testing and Evaluation of Service Oriented Systems, International Journal of Engineering Research and Application.

[5] Hirzalla, M., Cleland-Huang, J., &Arsanjani, A. (2009) A metrics Suite for Evaluating Flexibility and Complexity in Service Oriented Architectures, ACM.

[6] Elhag, M, A., &Mohamad, R. (2014). Metrics for Evaluating the Quality of Service Oriented Design, IEEE.

[7] Sharma, N., Bajpai, A., &Litoriya, R. (2012) Software Effort Estimation,International Journal of Computer Science and Applications.

[8] Zhang, Q., &Li, X. (2009). Complexity metrics for Service-Oriented Systems, Second International Symposium on Knowledge Acquisition and Modeling.

[9] Albrecht, A.J., & Gaffney, G.E. (1983) Software Function, Source lines of Codes, and Development Effort Prediction, A Software Science Validation, IEEE Trans Software Engineering.

[10] Munialo, S.W., &Muketha, G.M. (2016) A Review of Agile Software Effort Estimation Method,International Journal of Computer Applications Technology & Research..

[11] Mahmoud, K., Ilahi, M., Ahmed & B., Ahmed, S. (2012) "Empirical Analysis of Function points in service oriented in Service oriented architecture (SOA) Applications", : Industrial Engineering letters.

[12] Chidamber, S, R., &Kemerer, C, F. (1998) Managerial Use of Metrics for Object-Oriented Software: An Exploratory Analysis, IEEE.

[13] McCabe, T, J. (1976) A Complexity Measure,IEEE.

[14] Amsden, J. (2010) Modeling with SoaML, the Service-Oriented Modeling Language. Part 1. Service Identification, IBM.

[15] COSMIC. (2010) Guideline for Sizing SOA Software, v1.0 : The Common Software Measurement International Consortium (COSMIC).

[16] COSMIC. (2015) Guideline for Sizing SOA Software, v4.0 : The Common Software Measurement International Consortium (COSMIC)

[17] Martino, & S.,Gravino, C. (2009) Estimating Web Application using COSMIC-FFP Method,International Journal of Computer and Applications.

[18] Li, Z., & Keung, J. (2010) Software Cost Estimation Framework for Service-Oriented Architecture Systems Using Divide-and-Conquer Approach,5th IEEE International Symposium on Service Oriented System Engineering.

[19] Li, Z.,& O'Brien, L. (2010) Towards Effort Estimation for Web Service Compositions Using Classification Metrics, IEEE

[20] Marsyahariani, N., Daud, N. & Kadir, W.M.N. (2014) Static and Dynamic Classification for SOA Structural Attributes Metrics,8th Malaysian Software Engineering Conference.

[21] Sharma, V., Shewandayn, B. &Bhukya N. (2017) Measuring Usability of Web services using Coupling Metrics,International Journal of Advanced Research in Basic Engineering Science and Technology.

[22] Muketha, G.M., Ghani, &A., Selamat, M. (2010). A Survey of Business Process Complexity Metrics, Information Technology Journal.

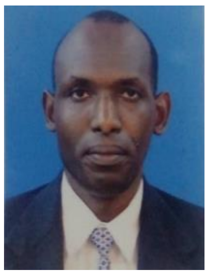
[23] Srinivasan, K. & Devi, T. (2014). Software Metrics Validation Methodologies in Software Engineering: International Journal of Software Engineering & Applications.

[24] Briand, L.C., Morasca, S., &Basili, C.H. (1991). Property – Based Software Engineering Measurement, IEEE Transactions on Software Engineering.

AUTHORS

Samson WanjalaMunialo is a an assistant lecturerinMeru University of Science and Technology, Kenya. He has BED. Degree from Catholic University of Eastern Africa, Kenya MSc Information Technology Management from University of Sunderland, UK and currently he is pursuing his PHD Information Technology at MasindeMuliro University of Science and Technology. His area of research includes software metrics, software effort estimation and IT project management.

Geoffrey Muchiri Muketha received the BSc degree in Information Science from Moi University in 1995, the MSc degree in Computer Science from Periyar University in 2004, and the PhD degree in Software Engineering from Universiti Putra Malaysia in 2011. He is Associate Professor and Dean of the School of Computing and Information Technology at Murang'a University of Technology, where he has taught and supervised both undergraduate and postgraduate students for many years. His research interests include software and business process metrics, software quality, verification and validation, empirical methods in software engineering, and component-based software engineering. He is a member of the International Association of Engineers (IAENG).



Dr. Kelvin Omienoisa Senior Lecturer in the Department of Information Technology and Informatics, School of Computing and Information Technology, Kaimosi Friends University College, Kenya. He holds a PhD in Information Systems of Jaramogi University of Science and Technology, MSc in Information Technology and BSc in Computer Science. His research interests are in performance evaluation, System optimization, security and parallel and distributed system.