# Measuring the Quality of Service Oriented Design

Renuka Sindhgatta, Bikram Sengupta, and Karthikeyan Ponnalagu

IBM India Research Laboratory
Bangalore, India
{renuka.sr,bsengupt,pkarthik}@in.ibm.com

**Abstract.** Service Oriented Architecture (SOA) has gained popularity as a design paradigm for realizing enterprise software systems through abstract units of functionality called services. While the key design principles of SOA have been discussed at length in the literature, much of the work is prescriptive in nature and do not explain how adherence to these principles can be quantitatively measured in practice. In some cases, metrics for a limited subset of SOA quality attributes have been proposed, but many of these measures have not been empirically validated on real-life SOA designs. In this paper, we take a deeper look at how the key SOA quality attributes of service cohesion, coupling, reusability, composability and granularity may be evaluated, based only on service design level information. We survey related work, adapt some of the well-known software design metrics to the SOA context and propose new measures where needed. These measures adhere to mathematical properties that characterize the quality attributes. We study their applicability on two real-life SOA design models from the insurance industry using a metrics computation tool integrated with an Eclipse-based service design environment. We believe that availability of these measures during SOA design will aid early detection of design flaws, allow different design options and trade-offs to be considered and support planning for development, testing and governance of the services.

**Keywords:** Service Design, Business Process Model, Service Design Principles, Metrics.

## 1 Introduction

Service Oriented Architecture (SOA) represents the natural continuum of increasing levels of abstraction in software engineering that has previously seen the emergence of object-oriented programming and component based development. SOA is characterized by a greater focus on identifying business-relevant functionality that may be exposed as *services* to consumers (end-user applications or other services), a higher-level of decoupling of interfaces and implementation, and a thrust on open standards-based protocols (e.g. Web Services) for realizing this vision.

The design of a service is guided by a set of principles that help in achieving the goals of SOA. These principles have been well-documented in the literature [6, 7, 19] and include notions of cohesion, coupling, reusability, composability, granularity, statelessness, autonomy, abstraction and so on. However, the principles are largely

L. Baresi, C.-H. Chi, and J. Suzuki (Eds.): ICSOC-ServiceWave 2009, LNCS 5900, pp. 485–499, 2009.

prescriptive in nature and there has been little work in defining how adherence to these principles may be quantitatively measured in practice. In some cases, metrics for a limited subset of SOA quality attributes have been proposed (e.g. [4, 5]), but most of these measures have not been empirically validated on real-life SOA designs. As a result, service design may proceed based on an informal or incomplete understanding of the principles, and without a sound measurement basis, could result in a flawed design. The generated services can provide all the functionality required by them and yet may not ultimately satisfy the design goals of SOA.

In this paper, we take a deeper look at how the key SOA quality attributes of service cohesion, coupling, reusability, composability and granularity may be evaluated, based only on service design level information. We review related work, adapt some of the well-known software design metrics to the SOA context and propose new measures where needed. We study their applicability on two real-life SOA design models from the insurance industry using a metrics computation tool integrated with an Eclipse-based service design environment. We also state the mathematical properties that the metrics adhere to (for lack of space, we do not include the proofs, which are straightforward). We believe that availability of these measures during SOA design will aid early detection of design flaws, allow different design options and trade-offs to be considered and support planning for development, testing and governance of services. The service consumer will also be capable of analyzing the quality of a service without having to analyze the details of the implementation (to which the consumer may not have access).

The rest of the paper is structured as follows. Section 2 sets the context by introducing the abstract service design model, case studies and tooling framework used in this paper. In Section 3, we define and evaluate a set of metrics for the SOA quality attributes of cohesion, coupling, reusability, composability, and granularity. Related work for each of the metrics is also discussed in detail and leveraged whenever possible. Section 4 presents directions for future research.

## 2 Setting the Context

We first describe the formal model and notation for service design that we use in this paper. Next, we introduce two large service designs in the Insurance Industry that we will use as running examples to compute and evaluate the metrics we propose. Finally, we briefly describe the service modeling environment on top of which our metrics computation tool has been built and our empirical studies conducted.

### 2.1 Model and Notations – Process, Service, Operations, Messages

To ensure common understanding of the metrics, we introduce the underlying service model and associated notations used in this paper. An enterprise adopting Service Oriented Architecture identifies a domain that needs to undergo SOA transformation.

- The business domain is supported by a set of business processes $P = \{p_1, p_2 \ldots p_P\}$.
- A set of services $S = \{s_1, s_2 \ldots s_S\}$ are identified and designed for automating the business process of the domain.

- A service $s \in S$ provides a set $O(s)$ of operations = $\{o_1, o_2, \ldots o_O\}$ and $|O(s)| = O$
- An operation $o \in O(s)$ has a set of input and output messages that are used as data containers between the service consumers and the service. A message and its constituent data types are derived from an information model of the domain. $M(o)$ is set of messages and data types for the operation $o$, The set of messages and constituent data types of all operations of a service $s$ is represented as $M(s) = \bigcup_{o \in O(s)} M(o)$.
- $S_{consumer}(s) = \{Sc_1, Sc_2 \ldots Sc_n\}$, represents a set of consumers of the service s.

### 2.2 Case Studies

**Insurance Application Architecture** (IAA): IAA [20] is a comprehensive set of insurance specific models that represent best practices in insurance. IAA describes the business of the insurer and includes process and information models of the domain. In recent years, a set of services have been designed to accelerate SOA adoption. In the rest of the paper, we refer to this design as ***ServiceDesignA***.

Insurance Property & Casualty Content Pack: IBM Websphere Industry content pack contains pre-built service-oriented architecture assets that are used to accelerate development of industry-specific business applications. The Insurance Property & Casualty Content Pack [21] for WebSphere Business Services Fabric focuses on property and casualty lines of business for insurance enterprises and provides a service design for the same. We refer to this design as ***ServiceDesignB***.

Table 1 gives a high-level summary of the design of the two experimental systems.

**Table 1.** Case Studies for Measuring Quality of Service Design

| Experimental System | # of services | # of operations | # of messages and types | # of Business Processes |
|---|---|---|---|---|
| ServiceDesignA | 110 | 622 | 3000 | 292 |
| ServiceDesignB | 83 | 286 | 794 | 53 |

### 2.3 Service Design and Metrics Computation Tool

Rational Software Architect (RSA) [22] provides a mature environment for designing SOA solutions and is built over the Eclipse platform supporting plug-in development. Our tool for metrics computation on service design is an RSA plug-in. A UML model of the service design is taken as input. Eclipse EMF APIs are used to extract data on each service e.g. operations, messages, data types and business processes. This data is used to compute the metrics through a metrics calculator. The metrics is stored along with each service design element and can be analyzed.

*We now move on to the main part of the paper – the definition and evaluation of a metric suite for different quality aspects of service-oriented design.*

## 3 Service Design Metrics

SOA design principles emphasize the attributes of coupling, cohesion, reusability, composability and granularity. Below, we briefly introduce each attribute and survey related work on measuring them, for procedural and OO systems. We also review the (limited) research in quantifying these attributes for service-oriented systems. Finally, we propose a set of metrics for measuring each attribute and study their applicability and usefulness on our example service design models.

### 3.1 Cohesion

For any system, cohesion measures the degree to which the elements of the system belong together [1]. The notion is generic enough to be applied to different types or levels of encapsulation e.g. a module, class, component, service etc., although how it is measured would have to be adapted to the context. Highly cohesive designs are desirable since they are easier to analyze and test, and provide better stability and changeability, which make the eventual systems more maintainable [10].

For procedural systems, various categories of module cohesion were proposed in [1] such as Coincidental (weakest), Logical, Temporal, Procedural, Communicational, Sequential and Functional (strongest). For Object-Oriented (OO) systems, a different set of categories was defined in [11]: Separable (weakest), Multifaced, Non-delegated, Concealed and Model (strongest). However, some of this categorization is subjective in nature. Bieman et. al [8] measure the functional cohesion of procedures by identifying common tokens that lie in the data slices of the procedure. Perhaps the most well-known effort at quantifying cohesion for OO systems is the LCOM (Lack of Cohesion in Methods) metric introduced by Chidamber and Kemerer that has multiple definitions and has undergone several refinements [3, 9].

For service-oriented systems, Perepletchikov et. al [5] categorizes cohesion on the basis of data, usage, sequence and implementation, defines measures for these and aggregates based on their average. Of the proposed measures, Service Interface Data Cohesion (SIDC), that identifies cohesion based on commonality of messages of the operations in terms of contained data types, will be reviewed in more detail below. None of the metrics have been empirically validated.

In the following, we first adapt two variants of the LCOM metric in the services context ($LCOS_1$, $LCOS_2$). The metrics are applied on our case studies and their drawbacks are analyzed. We propose a new metric for measuring service cohesion (SFCI) and evaluate its performance. Finally, the properties of SFCI are discussed.

**Lack of Cohesion of Service Operations (LCOS)**

LCOM has been widely used as a measure of cohesiveness in OO systems. For each class, the methods that operate on the same attributes are considered cohesive. In the context of services, there are no service attributes but messages become relevant as operations use these to execute the business functionality. Service operations that use common messages or their constituent data types can be considered cohesive. Service messages typically represent business entities or artifacts and hence operations on the same business entity or artifact are functionally related. We evaluate LCOM

definitions and redefine them for services. The definition is based on two widely used LCOM metrics [3, 9 ].

**$LCOS_1$** is based on the [3] where pairs of operations on the same set of messages are identified and considered cohesive; similarly, pairs of operations that do not contain similar messages are considered non-cohesive.

For a service *s* with operations *O(s)*, let $M(o_i)$ be the set of messages (and data types) used by operation $o_i \in O(s)$. Let,

$P(s) = \{ ( M(o_i), M(o_j) ) \mid M(o_i) \cap M(o_j) = \emptyset, o_i, o_j \in O(s) \}$ and

$Q(s) = \{ ( M(o_i), M(o_j) ) \} \mid M(o_i) \cap M(o_j) \neq \emptyset, o_i, o_j \in O(s) \}$, then

$LCOS_1(s) = |P(s)| - |Q(s)|$ *if* $|P(s)| > |Q(s)|$

$= 0$ *if* $|P(s)| < |Q(s)|$

As the above definition indicates, $LCOS_1$ is not normalized, similar to the original LCOM metric [6]. $LCOS_1$ is 0 (strong cohesion) when the number of operation pairs that share messages *(Q(s)),* is more than the number of pairs that do not *(P(s)).* Otherwise, the difference between the numbers is taken as the lack of cohesion measure.

**$LCOS_2$** is based on the [9]. The number of operations using a message *m* can be defined *as* $\mu(m)$ where $m \in M(s)$.

$$LCOS_2(s) = \frac{\left( \frac{1}{|M(s)|} \sum_{m \in M(s)} \mu(m) \right) - |O(s)|}{1 - |O(s)|}$$

$LCOS_2$ is bound between 0 and 1. If each operation uses all the messages ③(m) = |O(s)| and hence $LCOS_2 = 0$. If each operation uses a distinct message, then the numerator reduces to 1-|O(s)| and so $LCOS_2 = 1$.

In practice, we have found both $LCOS_1$ and $LCOS_2$ to suffer from some drawbacks when applied to service oriented systems. Apart from its lack of normalization, the discriminating power of $LCOS_1$ is low, and most services tend to be classified as highly cohesive. On the other hand, $LCOS_2$ tends to increase sharply with increase in the number of operations, and most services appear as lacking cohesion. This is because, with an increasing number of operations, it becomes very unlikely that each operation will require the same set of (all) messages, although they may still contain some core data types that are relevant to the service functionality and may thus be argued to be functionally cohesive. These observations motivated us to define the Service Functional Cohesion Index (SFCI) defined below.

**Service Functional Cohesion Index (SFCI)**

This metric defines the functional cohesion of the operations of the service based on the commonality of the *key* message(s) the operations use to perform the required functionality. As above, if the number of operations using a message *m* is $\mu(m)$ where $m \in M(s)$, and $|O(s)| > 0$, then

$$SFCI\ (s) = \frac{\max(\mu(m))}{|O(s)|}$$

*We define SFCI(s) to be 0 when s contains no operations.* In SFCI(s), we focus on the contained data types that defines the message that is most widely used across all the operations – the fraction of operations using this common message and types returns *SFCI*. The value of this metric is always between 0 (non-cohesive) and 1 (highly cohesive). The service is perfectly cohesive if all the operations use one common message – the intuition here is that *a cohesive service typically operates on a small set of key business objects (messages) relevant to that service,* so these objects should appear in most of its operations. But the operations may also need other messages as inputs to operate on the key objects, and these types can very well differ based on the nature of the operation. As our empirical studies will show, this metric is better indicative of the cohesion of service operations when compared to $LCOS_1$ and $LCOS_2$ and remains stable with increase in number of operations. To compute the above metric in practice, we recommend filtering out utility data types that are also part of the messages since otherwise, unrelated operations may appear cohesive. The classification of data types into utility and business-relevant types may be done by a domain expert. Utility data types (including those representing primitive types) usually appear in many/most operations, often across unrelated services, hence we may automatically identify potential utility data types based on their usage count, for validation and filtering by domain experts.

The Service Interface Data Coupling (SIDC) metric defined in [5] also considers common data types of messages across operations to measure service cohesion. However, like $LCOS_2$, cohesion is high in SIDC only when all operations have the messages with same data types. Also, the metric, which is defined as the ratio of two unrelated terms (the number of operations having the similar messages and the total number of messages) has not been normalized to range between 0 and 1. Finally, the metric has not been empirically evaluated.

**Measuring and Evaluating Cohesion Metrics**

We have evaluated $LCOS_1$, $LCOS_2$ and SFCI metrics on ServiceDesignA and ServiceDesignB, and the results are shown in Figure 1.Since $LCOS_1$ and $LCOS_2$ indicate lack of cohesion while SFCI measures cohesion; we plot $LCOS_1$, $LCOS_2$ and (1-SFCI). Along the X-axis, we have ordered the services in terms of their increasing number of operations.

In ServiceDesignA $LCOS_1$ indicates a value of 0 for all but 2 services, while in ServiceDesignB, it is 0 for all the services. Thus all services are deemed highly cohesive and are indistinguishable in this respect. Conversely, $LCOS_2$ displays a strong correlation with the number of operations, and cohesion is very low for all services with more than 5 operations. On the other hand, **the plot of SFCI shows better discriminating power compared to $LCOS_1$ and it remains stable as the number of operations increases, unlike $LCOS_2$.** To validate that SFCI is more meaningful as a cohesion metric than $LCOS_2$, we investigated a service PolicyAdministration having 9 operations, with $LCOS_2$ indicating *lack of cohesion of 0.85* and SFCI indicating *cohesion of 0.89*, which are very conflicting values. We found that all the 9 operations in PolicyAdministration are related to aspects of policy, and 8 of the 9 operations

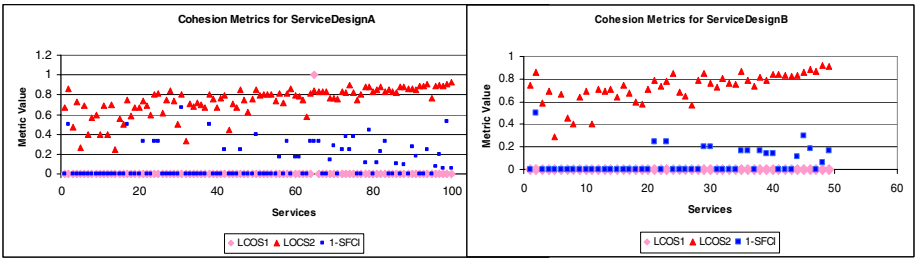



**Fig. 1.** Cohesion Metrics for ServiceDesignA and ServiceDesignB

process a business object called InsurancePolicy, hence from the design perspective, the service appears highly cohesive, as determined by SFCI, and the value of $LCOS_2$ appears misleading. We also reviewed a service with the lowest SFCI metric in ServiceDesignA. The service, LifePolicyManager has 19 operations dealing with different aspects such as terminating agreement, surrendering policy or requesting a loan, which could be refactored as multiple services. Note that there are several utility types that are defined to invoke an operation – e.g. RequestHeader, ResponseHeader and BusinessObject in ServicceDesignA. We filtered these types while computing the SFCI. It is seen that about 70% of the services in ServiceDesignA have an SFCI > 0.8. ServiceDesignB has 80% of the services with cohesion > 0.8. Thus both designs are very cohesive.

**Validation of Cohesion Metrics**

We verify the properties satisfied by the cohesion metric SFCI using the Properties based software engineering measurement framework [2]. SFCI is *not negative* and is *normalized* between 0 and 1 (*Non-negativity, Normalization*). SFCI is null when there are no messages or operations of a service (*Null Value*). SFCI is *monotonic* and does not reduce when more number of operations use some common messages. By adding more relationships between the messages and operations, $\mu$(m) increases and hence the cohesion of the service cannot decrease (*Monotonicity*). SFCI of a service obtained by putting together two unrelated services (having disjoint message sets) cannot be more than the SFCI of either service (*Cohesive Service*).

### 3.2 Coupling

Coupling measures the strength of association or dependence between systems. Loosely coupled systems are easier to maintain [10], since a change in one system entity will have less impact on other entities. They are also easier to comprehend, reuse and test. Low coupling and high cohesion are thus fundamental to the design of any software system, including those that are service-oriented.

The concept of coupling was originally studied for procedural systems and classified into different types of coupling such as *Content(highest),Data, Control, Messages(lowest)* coupling [1]. For OO systems, additional complexities in coupling introduced by inheritance, polymorphism etc. have been studied and a number of coupling frameworks have been proposed [11, 12]. Two well-known metrics for

OO coupling are *Coupling Between Objects (CBO)* and *Response for a Class (RFC)* [3]. CBO for a class is the count of the number of classes to which it is coupled – i.e. methods of one class use methods or instance variables of another. RFC for a class is the set of all methods that may be invoked in response to the invocation of a method in the class. In the context of service-oriented systems, [4] defines 8 types of coupling metrics. These metrics mostly relate to service implementation elements, assumes different weight factors for the relationships between elements, and makes many fine-grained distinctions between the types of dependencies. The aggregate forms of these metrics are used to define coupling at the service level. While the work is very detailed, the measures have not been empirically evaluated. Unlike [4], we define coupling measures assuming the availability of only service design level information. The focus is on defining a small set of metrics that are easy for the service designer to act on and the service consumer to comprehend. Our approach has been motivated by the fact that coupling as a property, has a tendency to generate a multitude of measures often without offering newer insight, as a study by Briand et al have shown for OO systems [13].

In a service-oriented design, we believe that there is a need to distinguish between 2 categories of coupling: the dependence of a service on other services, and its dependence on *messages*. The dependence of one service on another has parallels with inter-class coupling, and the OO metrics of CBO and RFC may be suitably adapted, as we show below. However, the dependence of a service on messages is a characteristic of the services domain. Unlike in OO where a class encapsulates data (class attributes) and also operations on that data, messages are not bound to a service; rather, they are treated as first-class entities in a service-oriented design approach, and are defined by data architects based on the information model containing all the business entities of the domain. Services encapsulate operations that refer to and update the state of the business messages, and thus become coupled to them – business object models may get independently updated, thereby necessitating changes to the service operations that process them. Accordingly, we define metrics for both service coupling (*SOCI, ISCI*) and message coupling *(SMCI),* below.

**Service Operational Coupling Index (SOCI)**

We analyze the dependence of a service on the operations of other services it uses for its functionality. Service Operational Coupling Index; SOCI can be represented as the number of operations of other services invoked by service *s*.

$$SOCI(s) = \left|\left\{o' \in s' \mid \exists_{o \in s} calls(o, o') \wedge s \neq s'\right\}\right|$$

*calls (o, o')* denotes a call made by operation *o* of *s* to operation *o'* of *s'*. This measure considers direct coupling only. We can further use a transitive closure of the *calls* relation to get a measure of indirect service operational coupling, which is denoted as $SOCI_{indirect}(s)$. SOCI is an adaptation of the OO metric Response for a Class (RFC) [3], in the services domain.

**Inter-Service Coupling Index (ISCI)**

Inter-Service Coupling Index (ISCI) is defined as the number of services invoked by a given service *s*.

$$ISCI(s) = \left|\{s' \mid \exists_{o \in s}, \exists_{o' \in s'}.calls(o,o') \wedge s \neq s'\}\right|$$

We can further use a transitive closure of the *calls* relation to get a measure of indirect inter-service coupling which is denoted as $ISCI_{indirect}(s)$. ISCI is similar in spirit to the OO metric of Coupling Between Objects (CBO) [3]. However CBO also includes dependencies on class attributes (in addition to methods), which is not relevant in the services context.

**Service Message Coupling Index (SMCI)**

SMCI measures the dependence of a service on the messages derived from the information model of the domain. These messages are those its operations receive as inputs, interpret and process, and those they need to produce as output, as declared in the interface. They also include messages the service needs to create in order to invoke operations in other services it is functionally dependent on. We represent SMCI as

$$SMCI\ (s) = \left| \bigcup\ M\ (o') \mid (o' \in\ s) \vee (\exists_{o' \in s'} \exists_{o \in s} calls\ (o,o') \wedge s \neq s') \right|$$

A low SMCI indicates less complexity for the service in interpreting and creating messages and less dependence on the domain information model. Note that *M(o)* includes all the constituent data types.

**Measuring and Evaluating Coupling Metrics**

The ISCI and SOCI metrics, evaluated on ServiceDesingnA, are shown in Fig.2 (a) Overall, the system has moderate levels of coupling and of the 110 services, 36 services (~ 33%) are coupled to other services, while the rest are atomic services that do not depend on other services for their functionality. For most services, the SOCI and ISCI metric are the same. This indicates that a service is dependent on another service for only one of its operations. Moreover, we have determined that the *Indirect* versions of these metrics do not bring in any additional coupling. The maximum value of ISCI is 4. The service OperationalRiskAssessment is coupled to other services as it analyzes risk by requesting information from 4 distinct services related to Customer, Policy, Agreement and Payment. In the case of SystemDesignB, all 83 services were atomic services. The design consists of utility services on which other services can be defined. Figure 2 (b) shows the SMCI metric for the services in ServiceDesignA and ServiceDesignB In general, ServiceDesingA has higher message coupling than ServiceDesignB, as seen from the figure.

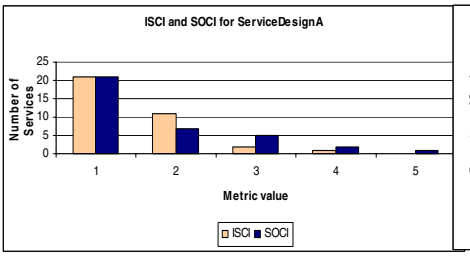
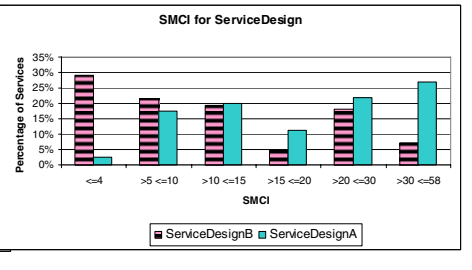



**Fig. 2(a).** ISCI, SOCI for ServiceDesignA




**2(b).** SMCI for ServiceDesignA and ServiceDesignB

**Validation of Coupling Metrics**

We now verify the properties of the coupling metrics [2]. The coupling metrics are *nonnegative* (*Nonnegativity*). ISCI, SOCI and SMCI are *null* if there are no coupled services or no messages for each of the service operations (*Null Value*). The metrics are *Monotonic* and *do not decrease* by adding more dependencies. SMCI may only increase if the number of messages of the service (or, in operations invoked by the service) increases (*Monotonicity*). The coupling of a service obtained by *merging two services* is less than or equal the sum of coupling of the two original services (*Merging of Services*). This is true for all the metrics. The coupling obtained by *merging two disjoint* services is equal to the sum of couplings of the two original services (*Disjoint Service Additivity*). Disjoint services are not consumers of each other, are coupled to different services and have disjoint message sets.

### 3.3 Reusability and Composability

We now discuss service reusability and composability, which are related concepts. Reusability is one of the key principles of service design. A service should ideally be designed for more that one service consumer. Service composability is a form of reusability. A service becomes a composition participant and can be reused along with other services to provide business functionality.

Reusability of an entity may be looked at from two perspectives: the characteristics of the entity that are predictors of reusability, and potential for future reuse of the entity based on usage that has already happened. The attributes of coupling and cohesion are generally good predictors of reusability. A service whose operations are cohesive and have less external dependencies will be more easily reusable. [14] computes *customizability*, *understandability* and *portability* metrics and uses them as predictors of reusability. Portability is measured in terms of the number of methods without parameters or return values. In [16], the average number of arguments per procedure is proposed as a measure of the understandability of the interface. For predicting reusability based on actual usage, contributions in terms of lines of code (LOC) [15] have been proposed for code assets. For OO systems, Depth of Inheritance (DIT) metric is used as a measure of reusability of a class [3]. However, neither of these metrics is relevant to services-oriented design, and we instead suggest measuring reusability based on use of the service by service consumers.

**Service Reuse Index**

The number of existing consumers of a service indicates the reusability of the service. At the service design level, these consumers may be other services coupled to this service or business processes where the service is used. We define Service Reuse Index as

$SRI(s) = |S_{consumer}(s)| = P + Q$, where

$P = \left|\left\{s' \mid \exists_{o \in s}, \exists_{o' \in s'}. calls(o', o) \wedge s \neq s'\right\}\right|$

$Q = \left|\left\{p \in P \mid s \in p\right\}\right|$

Similarly, we may define an Operation Reuse Index (ORI) for an operation as the number of consumers of that operation across services and business processes.

Sometimes the reuse of a service is due to the reuse of one or few of its operations – ORI helps identify those important operations of the service.

While SRI predicts future reuse based on existing usage of a service, service reuse potential based on interface understandability (along the lines of component understandability [16]) may be defined in terms of the complexity of the interface. The interface of a service is complex when it contains a high number of operations and messages, hence |O(s)| and |M(s)| may be used as indicators of understandability, with lower values implying better understandability (thereby higher reuse potential). However, proving the value of such measures for reuse (i.e. being able to link actual usage to better understandability) is difficult and higher interface complexity often means more reuse opportunities, as our empirical studies reveal below.

**Service Composability Index (SCOMP)**

A composable service is designed to participate as an effective member of multiple compositions. We define service composability considering the compositions in which the service is a composition participant and the number of distinct composition participants which succeed or precede the service. *Neighbors(s, p)* returns the set of services which are neighbors (immediate predecessors and successors) of *s* in business process *p*. We define:

$$SCOMP(s) = \left| \bigcup_{p \in P} Neighbors(s, p) \right|$$

We may also extend this definition to include other services that may not be immediate successors or predecessors of *s* but are participants of the same composition and would be present in the control flow of the composition. The composability of *s* with these services may be weighed by the inverse of its distance from *s* in these compositions (more distant is the neighbor, less is the composability).

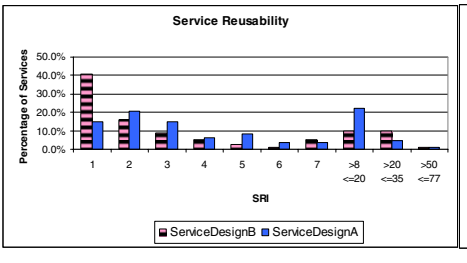
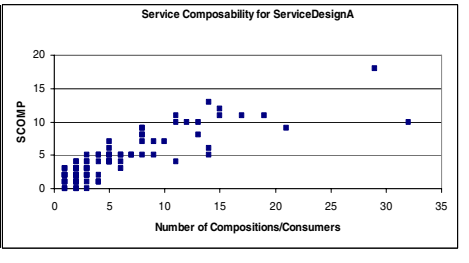



**Fig. 3(a).** SRI for ServiceDesignA and ServiceDesignB




**3(b).** SCOMP for ServiceDesignA

**Measuring and Evaluating Reusability and Composability**

Figure 3(a) shows the percentage of services having a certain number of consumers. There are some instances of high reuse e.g. in ServiceDesignA, there is one service 'PartyNotification' having 77 consumers. Similarly, in ServiceDesignB, there are 4 services that have >30 service consumers, but there is also a significant percentage of services with very few consumers. We evaluate the operation reuse index of the PartyNotification service. There is one operation that is highly reused as compared to the

others – notifyParty with 36 consumers. A change to this operation would have a high impact on the consumers of PartyNotification. SCOMP(s) for all the services of ServiceDesignA is measured as shown in Figure 3(b). As the number of compositions in which a service occurs increases, the number of distinct composition participants generally increases as well, and hence SCOMP increases. This correlation can be seen in the figure. This plot does not include the Party notification service that is used in 77 compositions and has SCOMP =32. We also found that for ServiceDesignA, service interface complexity (|M(s)| and |O(s)|) has a positive correlation of > 0.5 with reusability: it seems that higher the complexity (arguably, lower the understandability), the larger is the scope of service functionality, and higher the number of consumers. While interface complexity/understandability is an issue that may concern consumers from outside the domain looking to use the service, it seems that within a domain it is the value and scope of the business functionality offered by a service that determines its reuse potential. We return to this issue when we discuss service granularity.

**Properties of Reusability and Composability Metrics**

Based on the inherent semantics of reusability and composability, we define a set of properties that their metrics should adhere to. The metrics cannot be negative (*Non-negativity*). They should be null where there are no consumers *(Null Value)*. Reusability of a service or an operation should not decrease by adding more service consumers; similarly, composability of a service should not decrease with more composition participants (*Monotonicity*). The reusability of a service obtained by merging two services is not greater than the sum of reusability of the two original services. This is also true for the composability metric. *(Merging of services*). It may be shown that *SRI* and *SCOMP* satisfy these properties.

### 3.4 Service Granularity

Granularity refers to the quantity of functionality encapsulated in a service. A coarse grained service would provide several distinct functions and would have a large number of consumers. As described in [6], granularity could be further classified as *capability granularity* and *data granularity*. Capability granularity refers to the functional scope of the service and data granularity refers to the amount of data that is transferred to provide the functionality. One of the indicators of the quantity of functionality in a service is its size. The number of operations of a service $|O(s)|$ and the number of messages used by the operations $|M(s)|$ can be indicative of the Service *Capability Granularity (SCG)* and Service *Data Granularity (SDG)* respectively, where higher values may indicate coarser granularity e.g. larger functional scope. However, a high |O(s)| can also result from decomposing coarser operations into multiple finer-grained operations that consumers need to call, hence there is a need to reason about service granularity also from the perspective of a business process where the service is used. If a service encodes many small units of capability, each exchanging small amounts of data, then complex business processes would need a large number of such services to be composed to yield the desired functionality – thus for a business process $p \in P$, the number of services involved (*Process Service Granularity or PSG(p)*) and number of operations invoked (*Process Operation Granularity or POG(p)*), may also indicate if the constituent services are of an acceptable granularity or not – too

many ( conversely, too few) services and operations constituting a business process may imply that the services in the design model are too fine grained ( or, too coarse grained), and that there is a need to re-factor the services to get the granularity right. This is also related to the service identification process of *top-down decomposition* proposed by many methods (e.g. [18]), where a complex business process is successively decomposed into sub-processes, which ultimately map to services. The *Depth of Process Decomposition (DPD)* – the number of levels to which the process was decomposed before services were identified, can be an indicator of the granularity of the derived services and operations, with services/operations identified at a greater depth likely to be of finer granularity. Also, with each decomposition step, the potential number of services (and/or the number of operations in a service) may increase, thereby showing up as higher values of *PSG, POG*, *SCG* etc. Thus, service and process granularity metrics may need to be reviewed together, to obtain greater insight on design granularity.

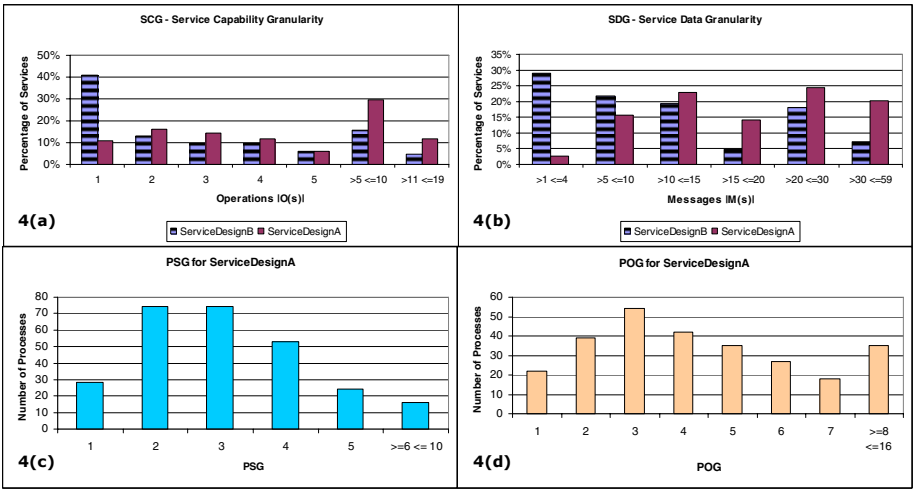



**Fig. 4.** Granularity Metrics for Service Design

## Measuring Granularity Metrics

We measure the granularity metrics for both the designs. As shown in Fig. 4, a large number of services in ServiceDesignB are fine grained with one operation and < 4 messages and types. In ServiceDesignA, there are many services with > 5 operations and >20 messages and types, which is indicative of coarser granularity of the services. *PSG(p)* of the processes of ServiceDesignA is shown in Figure 4(c ). There are about 20 processes that have one single service and invoke one operation as *POC(p)* =1. This indicates that the services used in the process are coarse grained. There are a few processes that involve around 10 services, and these may be explored to check if their granularity is too fine, but that is unlikely to be the case given that no process requires more than 16 operations. The DPD of the processes that we considered for the design

is 1 or 2, which suggests that processes were not overly decomposed to obtain services, and the rest of the metrics seem to confirm this.

**Properties of Granularity Metrics**
We validate the granularity metrics against the mathematical properties of size, as the number of services, operations, and messages are size metrics. The granularity of a service and a process is *nonnegative (Nonnegativity)*. The granularity of a service/process is *null* if it does not have any operations (*Null Value*). The granularity of a service obtained by *merging two disjoint services* is equal to the sum of the granularity of the original services having different messages and operations (*Disjoint Service Additivity*).

## 4 Discussions and Future Work

In this paper, we have proposed and evaluated a metrics suite for measuring the quality of service design along well-known design principles. The strengths and limitations of some of these metrics were discussed, and we have presented the results of measuring these metrics on two large SOA solution designs in the Insurance domain. Apart from conducting more empirical studies (with service designs from other domains), there are two tracks along which we are extending this work:

**Additional Service Design Qualities:** Some of the key service principles of abstraction, autonomy and statelessness have not been covered in this paper. These aspects of a service may require additional inputs that need to be defined during the design of services. For example, we are exploring WSDL-S [17] to see how such specifications may be analyzed to gain more quality insights.

**Design Analysis:** We have defined and analyzed the metrics independently. However, the principles are related, and often the same metric can be indicative of multiple design aspects, as we have seen (e.g. |M(s)| can be used to study coupling as well as granularity). In a large solution design, there are requirements to address multiple quality aspects of a solution, and these often involve trade-offs. The design would also need to account for the non-functional requirements such as governance and performance. A more comprehensive analysis of the design, that would allow users to prioritize design attributes and would propose design alternatives that best meet the business needs, is an important direction that we intend to explore.

## References

1. Stevens, W., Myers, G., Constantine, L.: Structured Design. IBM Systems J. 13, 115–139 (1974)
2. Briand, L.C., Morasca, S., Basili, V.R.: Property-Based Software Engineering Measurement. IEEE Trans. Software Eng. 22(1), 68–85 (1996)
3. Chidamber, S.R., Kemerer, C.F.: A Metrics Suite for Object Oriented Design. IEEE Trans. Software Eng. 20(6), 476–493 (1994)

4. Perepletchikov, M., Ryan, C., Frampton, K., Tari, Z.: Coupling Metrics for Predicting Maintainability in Service-Oriented Designs. In: Software Engineering Conference, ASWEC 2007, pp. 329–340 (2007)
5. Perepletchikov, M., Ryan, C., Frampton, K.: Cohesion Metrics for Predicting Maintainability of Service-Oriented Software. In: Seventh International Conference on Quality Software, pp. 328–335 (2007)
6. Erl, T.: SOA, Principles of Service Design. Prentice Hall, Englewood Cliffs (2007)
7. Artus, D.J.N.: SOA realization: Service design principles, http://www.ibm.com/developerworks/webservices/library/ws-soa-design/
8. Bieman, J., Ott, L.M.: Measuring Functional Cohesion. IEEE Transactions on Software Engineering 20(8), 644–657 (1994)
9. Henderson-Sellers, B.: Object-Oriented Metrics: Measures of Complexity. Prentice Hall, Englewood Cliffs (1996)
10. ISO/IEC, ISO/IEC 9126-1:2001 Software Engineering Product Quality – Quality Model, International Standards Organization, Geneva (2001)
11. Eder, J., Kappel, G., Schrefl, M.: Coupling and Cohesion in Object-Oriented Systems. In: ACM Conference on Information and Knowledge Management, CIKM (1992)
12. Briand, L.C., Daly, J., et al.: A Unified Framework for Coupling Measurement in Object-Oriented Systems. IEEE Transactions on Software Engineering 25(1), 91–121 (1999)
13. Briand, L.C., Daly, J., et al.: A Comprehensive Empirical Validation of Design Measures for Object-Oriented Systems. In: 5th International Software Metrics Symposium (1998)
14. Washizaki, H., Yamamoto, H., Fukazawa, Y.: A Metrics Suite for Measuring Reusability of Software Components. IEEE Metrics (2003)
15. Poulin, J., Caruso, J.: A Reuse Metric and Return on Investment Model. In: Advances in Software Reuse: Proceedings of Second International Workshop on Software Reusability, pp. 152–166 (1993)
16. Boxall, M., Araban, S.: Interface Metrics for Reusability Analysis of Components. In: Australian Software Engineering Conference, ASWEC (2004)
17. Web Service Semantics – WSDL-S, http://www.w3.org/Submission/WSDL-S/
18. Arsanjani, A.: Service-Oriented Modeling and Architecture, http://www.ibm.com/developerworks/library/ws-soa-design1/
19. Reddy, V., Dubey, A., Lakshmanan, S., et al.: Evaluation of Legacy Assets in the Context of Migration to SOA. Software Quality Journal 17(1), 51–63 (2009)
20. Huschens, J., Rumpold-Preining, M.: IBM Insurance Application Architecture (IAA) – An Overview of the Insurance Business Architecture. In: Handbook on Architectures of Information Systems, pp. 669–692. Springer, Heidelberg (1998)
21. IBM Insurance Property and Casualty Content Pack: http://www-01.ibm.com/support/docview.wss?rs=36&context=SSAK4R&dc=D400&uid=swg24020937&loc=en_US&cs=UTF-8&lang=en&rss=ct36websphere
22. IBM RSA: http://www-01.ibm.com/software/awdtools/architect/swarchitect/