An Evaluation on Developer's Perception of XML Schema Complexity Metrics for Web Services

Marco Crasso[1,2,3], Cristian Mateos[1,2,3], José Luis Ordiales Coscia[2], Alejandro Zunino[1,2,3], and Sanjay Misra[4]

[1] ISISTAN Research Institute.
[2] UNICEN University.
[3] Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET).
[4] Department of Computer Engineering, Atilim University, Ankara, Turkey

Abstract. Undoubtedly, the Service-Oriented Computing (SOC) is not an incipient computing paradigm anymore, while Web Services technologies is now a very mature stack of technologies. Both have been steadily gaining maturity as their adoption in the software industry grew. Accordingly, several metric suites for assessing different quality attributes of Web Services have been recently proposed. In particular, researchers have focused on measuring services interfaces descriptions, which like any other software artifact, have a measurable size, complexity and quality. This paper presents a study that assesses human perception of some recent services interfaces complexity metrics (Basci and Misra's metrics suite). Empirical evidence suggests that a service interface that it is not complex for a software application, in terms of time and space required to analyze it, will not be necessarily well designed, in terms of best practices for designing Web Services. A Likert-based questionnaire was used to gather individuals opinions about this topic.

Keywords: SERVICE-ORIENTED COMPUTING; WEB SERVICES; WEB SERVICE UNDERSTANDABILITY; WEB SERVICE COMPLEXITY; HUMAN PERCEPTION.

1 Introduction

Service-Oriented Computing (SOC) is a recent paradigm that allows developers to build new software by composing loosely coupled pieces of existing software, or *services* [1]. Services are usually provided by *providers*, who only expose services interfaces to the outer world, hiding technological details as much as possible. By means of these interfaces, potential *consumers* can determine what a service (functionally) offers and remotely invoke it from their applications. Consequently, SOC promotes not only code reuse but also process reuse, in the sense that the life cycle of invoked third parties services does not depend on consumers.

The high availability of broadband and ubiquitous connections nowadays allows users to reach the Internet from everywhere and at every time, enabling in turn the invocation of network-accessible services from within new software. In fact, the number of real published services has increased in the last few years. This has generated a global scale marketplace of services where providers offer their services interfaces, and consumers may invoke them regardless geographical aspects through the Internet [2] to

ensure ubiquity. Therefore, services are implemented using standard Web-enabled languages and protocols, and thus are called *Web Services*. Web Services is on the other hand the most common technological materialization of SOC, and find their application in diverse contexts such as migrating legacy systems to modern platforms [3] or exposing remotely-accessible information to smartphones [4].

Regardless the context of usage, much of the success of an individual Web Service depends on the quality of its interface, because in practice this is the only information source consumers have available when reasoning about the functionality offered by the service [5]. Moreover, even when many approaches that aim at simplifying the task of finding appropriate services exists [6], is the user who decides which service to select from a list of potential candidates. This decision unavoidably requires to manually inspect the description of the candidate services interfaces. Web Services interfaces are specified by using the WSDL, an XML-based format for describing a service as a set of operations, which can be invoked via message exchange.

Like any other software artifact, a WSDL document has many measurable attributes [7]. For example, [8] proposes a catalog of common bad practices found in WSDL documents, whereas in [7] and [9] two suites of metrics to assess the complexity of Web Services interfaces are proposed. There are almost two definitions of what it means for a WSDL document to be *complex*. One definition considers the execution time –time complexity– and space usage –spatial complexity– required by a software application to inspect and interpret a WSDL document. For instance, when automatically building a service *proxy* from a WSDL document, the complexity of a WSDL documents is directly related to the time and space needed to build the proxy. Furthermore, Kearney and his colleagues [10] alternatively state that complexity is defined "*by the difficulty of performing tasks such as coding, debugging, testing, or modifying the software*". Under this definition, complexity relates to how complex is for humans to inspect WSDL documents.

Previous research works have emphasized on measuring some non-functional concerns associated with interfaces in WSDL [8,7,9]. These concerns, specially complexity, have been found to be related to interface understandability, i.e. under Kearney's complexity definition, and consequently some WSDL-level metrics have been proposed. Based on these catalogs, service developers can modify and improve their WSDL documents until desired metrics values and therefore certain understandability levels are met. Therefore, it is really important for services providers to consider these WSDL-level metrics in order to control WSDL documents attributes.

This paper presents an evaluation of developers' perception of the core complexity metric described in [9], namely the Data Weight (DW) metric. The motivation of this study is that it is clear that DW reflects the time and space complexity of a data-type definition, i.e. DW is aligned with the first definition of complexity, whereas other subjective aspects of a WSDL document that impact on cognitive complexity, may not be so clearly reflected by DW. For instance, by definition of DW a definition having restrictions, attributes and extensions, which are typical constructors found in WSDL documents that allow developers to build well structured data-types, will be consider more complex than a primitive data-type, e.g. a string. In this paper we provide empirical evidence showing that the participants of our experiment perceive that WSDL

documents having well structured data-types have higher DW values, which means that there is a trade-off between the "weightlessness" of data-type definitions and adopting well-design practices for defining WSDL documents data-types, as the one described in [11].

The rest of the paper is organized as follows. Section 2 reviews related work focused on quality metrics for services interfaces and services interfaces improvement. The detailed experimental results are presented in Section 3. Section 4 concludes the paper and presents future work directions.

2 Background and related work

Web Services interfaces are described using WSDL, a language that allows providers to describe two main aspects of a service, namely what it does (i.e., its functionality) and how to invoke it (i.e., its binding-related information). Consumers use the former part to match external services against their needs, and the latter part to actually interact with the selected service. With WSDL, service functionality is described as a *port-type* $W = \{O_0(I_0, R_0), .., O_N(I_N, R_N)\}$, which lists one or more operations O_i that exchange input and return messages I_i and R_i, respectively. Port-types, operations and messages are labeled with unique names, and optionally they might contain some comments.

Messages consist of *parts* that transport data between providers and consumers of services, and vice-versa. Exchanged data is represented by using data-type definitions expressed in XML Schema Definition (XSD), a language to define the structure of an XML construct. XSD offers constructors for defining simple types, restrictions, and mechanisms to define complex constructs. XSD code might be included in a WSDL document using the *types* element, but alternatively it might be put into a separate file and imported from the WSDL document or external WSDL documents so as to achieve type reuse. Therefore, it is commonly said that Web Services data-types are specified in XSD.

There have been different research efforts to measure quality in WSDL-described services interfaces, but also to improve it, though in a comparative smaller quantity. The next subsections summarize related works.

2.1 Quality metrics for services interfaces descriptions

Previous research has emphasized on the importance of services interfaces and more specifically their non-functional concerns. The work of [8] identifies a suite of common bad practices or anti-patterns found in services interfaces, which impact on the understandability and discoverability of described services. Here, understandability is the ability of a service interface description of being self-explanatory, i.e., no extra software artifact apart from a WSDL is needed to understand the functional purpose of a service.

Discoverability refers to the ability of a service of being easily located when consumers inquiry the registry where the service is stored. The suite consists of eight bad practices that frequently occur in a corpus of public WSDL documents. To assess the understandability and discoverability of a WSDL document, one could account bad

practices occurrences because the fewer the occurrences are, the better the resulting WSDL document is. The authors then offer a tool called Anti-patterns Detector [12], which automatically computes the proposed metrics based on an input WSDL document.

[7] describes a metrics suite that consists of different kinds of metrics, ranging from common size measurements like lines of code and number of statements, to metrics for measuring the complexity and quality of services interfaces. All the involved metrics can be computed from a service interface in WSDL, since the metric suite is purely based on WSDL schema elements occurrences. The most relevant complexity metrics included in the suite are Interface Data Complexity, Interface Relation Complexity, Interface Format Complexity, Interface Structure Complexity, Data Flow Complexity (Elshof's Metric), and Language Complexity (Halstead's Metric). Moreover, the proposed quality metrics are Modularity, Adaptability, Reusability, Testability, Portability, and Conformity. Metrics results are expressed as a real coefficient in [0,1]. For complexity metrics, a value in [0-0.4) indicates low complexity, in [0.4-0.6) indicates medium complexity, in [0.6-0.8) indicates high complexity and in [0.8-1] indicates that the service is not well designed at all. Instead, for quality metrics [0-0.2) indicates no quality at all, [0.2-0.4) indicates low quality, [0.4-0.6) indicates medium quality, [0.6-0.8) indicates high quality, and [0.8-1.0] indicates very high quality.

Regarding services interfaces complexity, Baski and Misra [9] present a metric suite (BM metrics suite) whose cornerstone is that the effort required to understand data sent to and from the interfaces of a service can be characterized by the structures of the messages that are used for exchanging and conveying the data. Basing on this statement, Baski and Misra define five metrics: Data Weight (DW), Distinct Message Count (DMC), Distinct Message Ratio (DMR), Message Entropy (ME) and Message Repetition Scale (MRS), which can also be computed from a service interface in WSDL, since this metric suite is purely based on WSDL and XML schema elements occurrences. Below, we further explain these metrics since this paper is based on them.

Data Weight metric The definition of the Data Weight (DW) metric computes the complexity of the data-types conveyed in services messages. For the sake of brevity, we will refer to the complexity of a message $C(m)$ as an indicator of the effort required to understand, extend, adapt, and test a message m by basing on its structure. $C(m)$ counts how many elements, complex types, restrictions and simple types are exchanged by messages parts, as it is further explained in [9]. Formally:

$$DW(wsdl) = \sum_{i=1}^{n_m} C(m_i) \tag{1}$$

, where n_m is the number of messages that the WSDL document exchanges. The DW metric is a positive integer. The bigger the DW of a WSDL document, the more complex its operations messages are. For the purposes of this paper, we have assumed n_m to consider only those messages that are linked to an offered operation of the WSDL document, thus the DW metric does not take into account dangling messages.

Distinct Message Count metric Distinct Message Count (DMC) metric can be defined as the number of distinct-structured messages represented by $[C(m), n_{args}]$ pairs, i.e., the complexity value $C(m)$ and total number of arguments n_{args} that the message contains [9].

To better illustrate the DMC metric, in Figure 1 two WSDL documents defining an operation for returning the weather report in a city are shown. The WSDL document on the left defines two operations (GetWeatherReportIn y GetWeatherReportOut) pointing to wrapper data-types encapsulating one argument (city and report, respectively). It is easy to see that both messages have the same complexity, and thus computing DMC would output $[C(GetWeatherReportIn), 1]$ and $[C(GetWeatherReportOut), 1]$ as the pairs. As a result, DMC is zero since there are not distinct pairs. On the other hand, by looking at the WSDL document on the right, the GetWeatherReportRequest data-type has now two arguments. The associated pairs are thus $[C(GetWeatherReportIn), 2]$ and $[C(GetWeatherReportOut), 1]$, resulting in $DMC = 2$.

```
...
<wsdl:types>
  <xsd:element name="GetWeatherReportRequest">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element maxOcurrs="1" minOcurrs="1"
                     name="city" type="xsd:string"/>
      <xsd:sequence>
    <xsd:complexType>
  </xsd:element>
  <xsd:element name="GetWeatherReportResponse">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element maxOcurrs="1" minOcurrs="1"
                     name="report" type="xsd:string"/>
      <xsd:sequence>
    <xsd:complexType>
  </xsd:element>
<wsdl:types>
<wsdl:message name="GetWeatherReportIn">
  <wsdl:part element="tns:GetWeatherReportRequest"
                     name="response"/>
</wsdl:message>
<wsdl:message name="GetWeatherReportOut">
  <wsdl:part element="tns:GetWeatherReportResponse"
                     rname="response"/>
</wsdl:message>
...
```

```
...
<wsdl:types>
  <xsd:element name="GetWeatherReportRequest">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element maxOcurrs="1" minOcurrs="1"
                     name="latitude" type="xsd:float"/>
        <xsd:element maxOcurrs="1" minOcurrs="1"
                     name="longitude" type="xsd:float"/>
      <xsd:sequence>
    <xsd:complexType>
  </xsd:element>
  <xsd:element name="GetWeatherReportResponse">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element maxOcurrs="1" minOcurrs="1"
                     name="report" type="xsd:string"/>
      <xsd:sequence>
    <xsd:complexType>
  </xsd:element>
<wsdl:types>
<wsdl:message name="GetWeatherReportIn">
  <wsdl:part element="tns:GetWeatherReportRequest"
                     name="response"/>
</wsdl:message>
<wsdl:message name="GetWeatherReportOut">
  <wsdl:part element="tns:GetWeatherReportResponse"
                     name="response"/>
</wsdl:message>
...
```

Fig. 1. DMC: Examples

Distinct Message Ratio metric The Distinct Message Ratio (DMR) metric complements DW by attenuating the impact of having different messages within a WSDL document with the same structure. As the number of similarly-structured messages increases the complexity of a WSDL document decreases, since it is easier to understand similarly-structured messages than structurally different ones as a result of gaining familiarity with repetitive messages [9]. Formally:

$$DMR(wsdl) = \frac{DMC(wsdl)}{n_m} \tag{2}$$

The DMR metric provides a number in the range given by $[0, 1]$, where 0 means that all defined messages are similarly-structured, and 1 means that all messages are dissimilar. Therefore, the WSDL documents of Figure 1 (left) and Figure 1 (right) have a DMR of $0/2 = 0$ and $2/2 = 1$, respectively.

Message Entropy metric The Message Entropy (ME) metric exploits the probability of similarly-structured messages to occur within a given WSDL document. Compared with the DMR metric, ME also bases on the fact that repetition of the same messages makes a developer more familiar with the WSDL document and results in ease of understandability, but ME provides better differentiation among WSDL documents in terms of complexity. Formally:

$$ME(wsdl) = \sum_{i=1}^{DMC(wsdl)} P(m_i) * (-log_2 P(m_i)) \quad (3)$$
$$P(m_i) = \frac{nom_i}{n_m}$$

, where nom_i is the number of occurrences of the i^{th} message, and $P(m_i)$ represents the probability that such a message occurs within the given WSDL document. The ME metric has values in the range $[0, log_2(n_m)]$. A low ME value shows that the messages are consistent in structure, which means that data complexity of a WSDL document is lower than that of other WSDLs having equal DMR values.

Message Repetition Scale metric The Message Repetition Scale (MRS) metric analyzes variety in structures of WSDL documents. MRS measures the consistency of messages by considering frequencies of $[C(m), n_{args}]$ pairs, as follows:

$$MRS(wsdl) = \sum_{i=1}^{DMC(wsdl)} \frac{nom_i^2}{n_m} \quad (4)$$

The possible values for MRS are in the range $[1, n_m]$. When comparing two or more WSDL documents, a higher MRS and lower ME show that the developer needs less effort to understand the messages structures due to the repetition of similarly-structured messages.

2.2 Approaches to improve services interfaces

As far as we know, two main approaches to improve services interfaces have been explored. One approach occurs at the deployment phase of services and WSDL documents. The other approach is called "early", since it deals with the improvement of interfaces during the implementation phase of the underlying services, i.e., prior to obtain their corresponding WSDL documents.

The work of [8] fits in the first approach mentioned, since it identifies WSDL bad practices and supplies guidelines to remedy these. A requirement inherent to applying

these guidelines is following *contract-first*, a method that encourages designers to first build the WSDL document of a service and then supply an implementation for it. In this context, the term contract refers to *technical contract* and *interface*, indistinctly. However, the most used method to build WSDL documents in the industry is *code-first*, which means that one first implements a service and then generates the corresponding WSDL document by automatically extracting and deriving the interface from the implemented code. This means that WSDL documents are not directly created by humans but are instead automatically derived via language-dependent tools, which essentially map source code to WSDL code.

With regard to the early approach, the idea is to anticipate potential quality problems in services interfaces. Conceptually, the approach is to identify refactoring operations for services implementations that help to avoid problems in services interfaces. The main hypothesis of this approach is the existence of statistical relationships between two groups of metrics, one at the service implementation level and another at the service interface level. This implies that at least one metric in the former group could be somehow "controlled" by software engineers attempting to obtain better WSDL documents, with respect to what is measured by the metrics that belong to the latter group. This also means that if a software engineer modifies his/her service implementation and in turn this produces a variation on implementation level metrics, this change will be propagated to services interfaces metrics, assuming that both groups of metrics are correlated. Thus, key to this approach is understanding how implementation level metrics relate to interface level ones.

The work presented in [13] studies the relationships between service implementation metrics and the occurrences of the bad practices investigated in [8], when such interfaces are built using the code-first method. This bad practices come with a number of metrics not to assess complexity but to measure service discoverability, i.e., how effective is the process of ranking or locating an individual service based on a meaningful user query that describes the desired functionality. Then, the authors gathered 6 classic OO metrics from several services implementations, namely Chidamber and Kemerer's [14] CBO (Coupling Between Objects), LCOM (Lack of Cohesion of Methods), WMC (Weighted Methods Per Class), RFC (Response for Class), plus the CAM (Cohesion Among Methods of Class) metric from the work of Bansiya and Davis [15] and the well-known lines of code (LOC) metric. Additionally, they gathered 5 ad-hoc metrics, namely TPC (Total Parameter Count), APC (Average Parameter Count), ATC (Abstract Type Count), VTC (Void Type Count), and EPM (Empty Parameters Methods). Then, the authors collected a data-set of publicly available code-first Web Services projects and analyzed the statistical relationship among metrics. Finally, the correlation analysis shows that some WSDL bad practices are strongly correlated with some implementation metrics.

As the reader can see, previous efforts present in the literature strongly suggest that employing correlation analysis among classical software engineering metrics and other metric suites is in the right direction towards predicting undesirable situations. In this sense, in [16] the authors analyze the correspondence between the metrics described in Section 2.1 and classical software engineering metrics. The achieved results show that there are statistically significant relationships among pairs of metrics. Accordingly, the

authors conclude that by monitoring several metrics, which are present at services implementations, the complexity, in particular the Data Weight (DW), of the target WSDL documents can be reduced [16].

Recent investigations show that the refactorings needed to reduce DW in WSDL documents affect other quality concerns, in particular the adoption of best design practices described in [11]. The experimental results showing these undesirable relationships are going to be explained in another paper, i.e., the explanation of the statistical correlations between DW and anti-patterns is out of the scope of this paper, but a few words about this phenomena are need to clearly explain the focus of this paper. The DW metric measures the complexity of a data-type in XSD, from the perspective of a software application that must interpret and process such data definition. For example, parses like Xerces, and xjc require more CPU processing for parsing bigger data-types definitions, in terms of lines of code. This is because DW increases as $C(m)$ counts how many elements, complex types, restrictions and simple types are exchanged by messages parts. However, not always a bigger data-type definition would be a bad design decision. In general, when using XSD for defining data-types, more lines of code means a more detailed definition, which in turn makes easier to represent a business object specifically. A contra-example of this situation is given when defining a data-type using general-purpose data-types, like the xsd:any, called "wild-card" in [17], which allows defining almost any XSD content in just one line of code: <element name="theName" type="xsd:any"/>. Understanding business-object represented by this data-type is impossible, unless natural language documentation accompanying the data-type definition provides enough information. On the other hand, as explained in [18] when defining wild-cards developers are breaking best design practices of WSDL documents [11].

Returning to the mentioned correlations between best design practices and the DW metric, this paper represents a step towards understanding why the refactorings needed to reduce DW may deteriorate the design quality of WSDL documents. In this sense, this paper analyzes human beings' opinions about the DW metric. This paper aims to answer whether there is a trade-off between achieving low DW values, i.e., a smaller complexity, and preserving the highest possible standards in terms of WSDL documents design quality.

3 Experimental results

We performed a controlled experiment with humans to assess how they perceive Data Weight (DW) metric values. The experiment involved 27 participants that were asked to complete a survey to collect their opinions. The survey was designed as a Likert's based questionnaire for determining to whether there is a trade-off between services data-types design quality and how complex a service description is for a computer application, i.e., the DW metric values.

The survey was developed in the context of the "Service-Oriented Computing"[5] course of the Systems Engineering at the Faculty of Exact Sciences (Department of Computer Science) of the UNICEN during 2012. The course has been offered since in

[5] http://www.exa.unicen.edu.ar/~cmateos/cos

2008, is optional, and its audience are last-year undergraduate students and postgraduate students (both master and doctoral programs) without knowledge on SOC concepts. The course requirements are excellent skills on programming and some experience with Java development. In the context of our experiment, this means that the participants could be regarded as software developers. After five lectures within one week of three hours each discussing the fundamentals of the SOC paradigm and enabling technologies the students were instructed to develop a code-first Web Service, then a contract-first Web Service, and finally an application that consumes the previous services. After that, the students were invited to complete the survey. The survey asks the participants to analyze different versions of a Web Service. Each version of the service has been chosen to present different levels of the DW metric along with data-types having different design quality. The design quality of each data-type was assessed in accordance to the catalog of best design practices of [11]. To do this, three Web Services specialist follow the conventions and recommendations of W3C XML Schema Definition 1.1[6].

The questionnaire has been designed as a set of 18 statements. For each statement, the participant can choose among 6 alternatives, being *totally agree*, *agree*, *somewhat agree, somewhat disagree, disagree* and *completely disagree.* The reason to choose among six alternatives is that by being a pair number there is not chance for neutral opinions. We have included in the questionnaire 12 positive statements and 6 negative ones. A positive statement is a statement that textually confirms the existence of the hypothesis that the survey is testing, e.g., the existence of a trade-off between DW metric values and services interfaces design quality. On the other hand, a negative statement is one that denies the existence of the mentioned trade-off. This is important to note, because each statement is associated with a numerical score, which is known as Likert's score. The reason to have both positive and negative statements is to include contingent statements. A contingent statement is employed for re-formulating a statement to achieve more confident about a participant's response.

The numerical score of a statement depends on the polarity of the statement. For positive statements the numerical score ranges from 5 (totally agree) to 0 (totally disagree), whereas negative statements are inversely scored, i.e., ranging from 0 (totally agree) to 5 (totally disagree).

We computed the Likert score per participant. The numerical score for a participant's questionnaire is calculated by summing the individual score of all his/her statements. Then, the bigger the numerical score of a participant, the stronger is his/her confident about the existence of the trade-off between DW metric values and services interfaces design quality. A score equals to 0 means that the participant strongly disagrees with the trade-off existence, but a 90 score means that the participant strongly agrees with trade-off existence. For readability purposes, we translated the Likert's scores from a [0,90] scale to a [0,100] scale.

Figure 2 shows the score histogram, where each bar contains the number of participants who had the same score. Figure 2 shows that nine participants did not achieve the same score as any other participant. As shown in the Figure, this situation corresponds to participants achieving extreme scores, i.e., the lowest or the highest scores. At the same time, four participants achieved the same score, as denoted by the highest

[6] XML Schema Definition (XSD), http://www.w3.org/TR/xmlschema11-1/

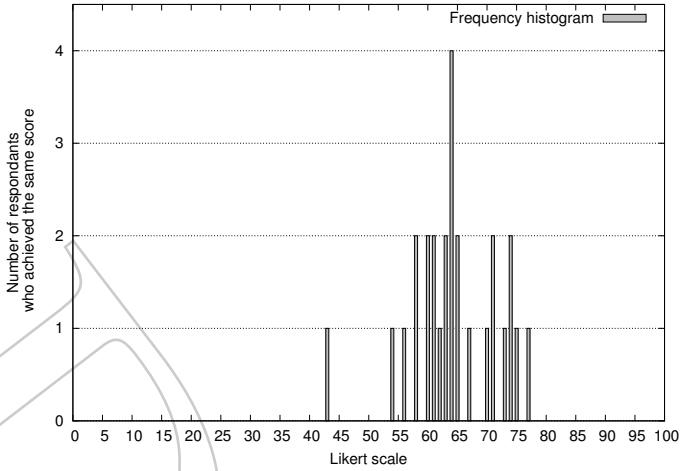

Fig. 2. Likert scale: Frequency of the scores.

bar of Figure 2, and they were positioned in the middle of the score scale and not in the extremes.

Figure 3 shows that by smoothing the frequency results using Bézier curves, they tended to a normal distribution with an average $\mu = 65$ and a standard deviation $\sigma = 7.5$. Then, 95.4% of the students scored between $[\mu - 2 * \sigma, \mu + 2 * \sigma]$. In other words, 25 students scored in the range of $[50, 80]$, which manifests the existence of a trade-off between DW metric values and WSDL design criteria.

In other words, the previous results provide empirical evidence showing that the surveyed humans perceives that for a given WSDL document having high DW metric values will not necessarily mean that the service interface is complex for them. On the other hand, the surveyed humans confirmed that a well designed service interface could have a higher DW metric value in its associated WSDL document than a poorly designed one.

The questionnaire included a statement designed to assess humans' perception of two aspects of a service interface that impact on the service description complexity. One aspect is the use of descriptive names for denoting WSDL elements. This aspect is important for the purposes of the experiment as the DW metric only considers the syntactic information of a WSDL document. On the other hand, the other aspect relates to arranging error information using special WSDL fault messages. This aspect is important because for the DW metric another WSDL message would increase service complexity, whereas in [11] the authors identify *"using a separated (extra) message for exchanging fault information"* as a best design practice for WSDL documents. Figure 4 shows that the average score for the participants was 2. The y-axis of the Figure also shows the statement text. As the reader can see, the participants did not share a strongly defined opinion about the usage of descriptive names and fault messages as the only drivers for service interface quality. Accordingly, more work should be done to analyze

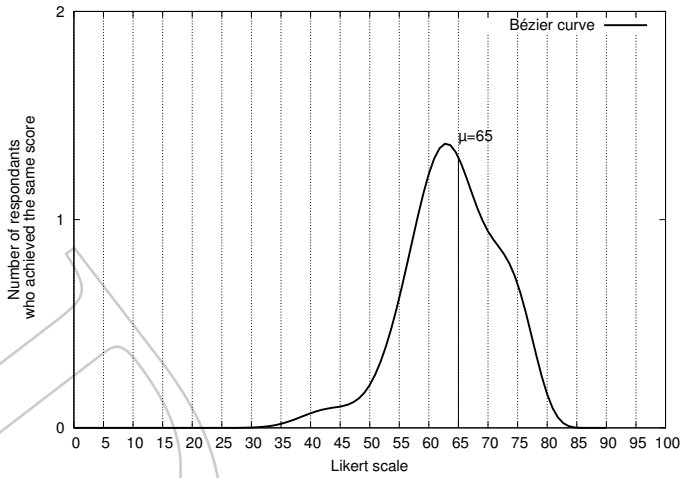

Fig. 3. Likert scale: Distribution of the scores.

the humans' perception of these concerns, in order to include them in the calculation of the DW metric.

4 Conclusions and Future Work

This papers states that the core metric of the metric suite described in [9] is very useful for assessing the complexity of WSDL documents, when we refer to complexity as the computational effort that a software application (e.g., a parser) must put to interpret and "consume" the document, but the metric may be revised and improved in order to use it to assess the design quality of such a document. Historically, there has been controversy between software engineering metrics that are based on syntactical constructions and those based on subjective aspects. For example, when T. McCabe discusses the effectiveness of physical size driven metrics and use as an example a 50 line program consisting of 25 consecutive "IF THEN" constructs, he shows that those metrics based on LOC might not provide an accurate overview of the program complexity. In this case, we have identified a similar situation in which a WSDL document having few lines of code will have a smaller DW than another having more code and potentially a better design, in terms of data-type definitions.

A major limitation of the presented study is the number of participants that have been surveyed, although some tendencies could be observed from their responses. In this sense, the study shows that the participants perceive the existence of a trade-off between the DW metric and the design quality of services interfaces descriptions. All in all, more work should be done in order to determine and possibly adjust the DW metric definition, or define a complementary metric. Moreover, the study has shown that such a complementary metric may consider two important aspects ignored by DW, namely data-types names and fault-messages.

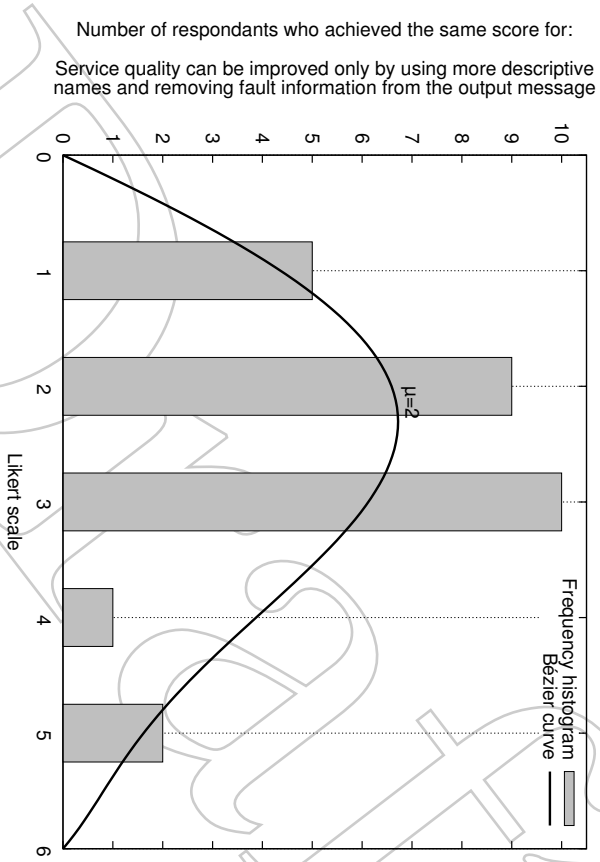

Fig. 4. Likert scale: Frequency of the scores.

We are planning to extend the present paper by including a statistical analysis that shows the correlation between the DW metric and WSDL best practices. This could be done by quantifying the level of compliance of WSDL documents with the WSDL design practices proposed in [11] and performing appropriate statistical analyses.

Acknowledgments

We acknowledge the financial support provided by ANPCyT (PAE-PICT 2007-02311).

References

1. Martin Bichler and Kwei-Jay Lin. Service-Oriented Computing. *Computer*, 39(3):99–101, 2006.
2. S. Wang, Q. Sun, H. Zou, and F. Yang. Reputation measure approach of Web Service for service selection. *IET Software*, 5(5):466–473, 2011.
3. Juan Manuel Rodriguez, Marco Crasso, Cristian Mateos, Alejandro Zunino, and Marcelo Campo. Bottom-up and top-down COBOL system migration to Web Services: An experience report. *IEEE Internet Computing*, 2011. To appear.
4. Guadalupe Ortiz and Alfonso García De Prado. Improving device-aware Web Services and their mobile clients through an aspect-oriented, model-driven approach. *Information and Software Technology*, 52(10):1080–1093, 2010.
5. Marco Crasso, Juan Manuel Rodriguez, Alejandro Zunino, and Marcelo Campo. Revising WSDL documents: Why and how. *Internet Computing*, 14(5):30–38, 2010.
6. Marco Crasso, Alejandro Zunino, and Marcelo Campo. A survey of approaches to Web Service discovery in Service-Oriented Architectures. *Journal of Database Management*, 22(1):103–134, 2011.
7. Harry M. Sneed. Measuring Web Service interfaces. In *12th IEEE International Symposium on Web Systems Evolution (WSE), 2010*, pages 111 –115, September 2010.
8. Juan Manuel Rodriguez, Marco Crasso, Alejandro Zunino, and Marcelo Campo. Improving Web Service descriptions for effective service discovery. *Science of Computer Programming*, 75(11):1001–1021, 2010.
9. D. Baski and S. Misra. Metrics suite for maintainability of extensible markup language Web Services. *IET Software*, 5(3):320–341, 2011.
10. Joseph P. Kearney, Robert L. Sedlmeyer, William B. Thompson, Michael A. Gray, and Michael A. Adler. Software complexity measurement. *Communications of the ACM*, 29(11):1044–1050, November 1986.
11. Juan Manuel Rodriguez, Marco Crasso, Cristian Mateos, and Alejandro Zunino. Best practices for describing, consuming, and discovering Web Services: a comprehensive toolset. *Software: Practice and Experience*, 2012.
12. Juan Manuel Rodriguez, Marco Crasso, Alejandro Zunino, and Marcelo Campo. Automatically detecting opportunities for web service descriptions improvement. In Wojciech Cellary and Elsa Estevez, editors, *Software Services for e-World*, IFIP Advances in Information and Communication Technology, pages 139–150, Boston, MA, USA, 2010. Springer.
13. Cristian Mateos, Marco Crasso, Alejandro Zunino, and José Luis Ordiales Coscia. Detecting WSDL bad practices in code-first Web Services. *International Journal of Web and Grid Services*, 7(4):357–387, 2011.
14. S. Chidamber and C. Kemerer. A metrics suite for Object Oriented design. *IEEE Transactions on Software Engineering*, 20(6):476–493, 1994.

15. Jagdish Bansiya and Carl G. Davis. A hierarchical model for Object-Oriented design quality assessment. *IEEE Transactions on Software Engineering*, 28:4–17, January 2002.
16. José Luis Ordiales Coscia, Marco Crasso, Cristian Mateos, Alejandro Zunino, and Sanjay Misra. Predicting Web Service maintainability via Object-Oriented metrics: A statistics-based approach. In Beniamino Murgante, Osvaldo Gervasi, Sanjay Misra, Nadia Nedjah, Ana Rocha, David Taniar, and Bernady Apduhan, editors, *Computational Science and Its Applications (ICCSA 2012), Salvador de Bahia, Brazil*, volume 7336 of *Lecture Notes in Computer Science*, pages 29–39. Springer Berlin / Heidelberg, 2012.
17. James Pasley. Avoid XML schema wildcards for Web Service interfaces. *IEEE Internet Computing*, 10:72–79, 2006.
18. Marco Crasso, Juan Manuel Rodriguez, Alejandro Zunino, and Marcelo Campo. Revising WSDL documents: Why and How. *IEEE Internet Computing*, 14(5):48–56, 2010.