Comparing Web Services with Other Software Components

Yijun Yu
y.yu@open.ac.uk
Computing Department
The Open University

Jianguo Lu
jlu@cs.uwindsor.ca
School of Computer Science
University of Windsor

Juan Fernandez-Ramil
j.f.ramil@open.ac.uk
Computing Department
The Open University

Phil Yuan
yuanf@uwindsor.ca
School of Computer Science
University of Windsor

Abstract. *Software metrics is vital for the management of software development, especially when a new technology is adopted and established practices have yet to be developed. As a kind of software components, web service technology has flourished and attracted a flurry of research activities. Despite the vast amount of research on mechanisms of web services, there have been little investigations of the overall nature of existing web services from a software component point of view. This paper is the first attempt to compare web services with other software components in terms of established metrics in software engineering, including object oriented metrics and interface metrics. In this study we conclude that there are statistical differences between the interface, variable name and other OO metrics when one compares a large sample web services with typical OO systems. The distributions obtained give insight into the typical characteristics of web services and can be used to identify candidates for wrapping into web services.*

Keywords: *web service, software metrics, interface, component, XML Schema*

1. Introduction

Software metrics is important to the management of software development, and is a mature field that has been studied for decades [17]. “The need for (software) metrics is particularly acute when an organization is adopting a new technology for which established practices have yet to be developed”[7]. Although web service technology has been adopted by major software vendors such as IBM, Microsoft, BEA, Oracle, Borland, etc., and resulted in a flurry of research activities, there are little studies on existing web services in terms of software metrics. Web services can be seen as software components where implementation details are hidden behind the interfaces. Many web service interfaces are automatically generated by wrapping existing software systems using a language-to-web service binding tool such as .Net frameworks, Apache Axis java2wsdl, etc. To understand the state-of-the-art of web services, there is a need to compare web services with other software components that offer similar functionality in terms of established software metrics.

Studies on web service metrics will benefit the development and management of web services as well as the research on it. As a developing and evolving technology, new standards are being proposed, existing standards such as SOAP (Simple Object Access Protocol) [4], UDDI (Universal Description, Discovery and Integration [18]), WSDL (Web Service Definition Language) [22] are constantly under revisions, researches are flourishing on web service modeling, discovery, composition, and verification. On the other hand, web service developers need to know the best practice or the “styles” of web service implementation, better based on statistics of the existing web services. All these research and development activities require the investigation of web services metrics.

Web services are software components where implementation details are hidden behind the interfaces. In their interface definitions, web services are self-descriptive software component whose data are expressed using XML Schemas, and whose interfaces are expressed using the Web Service Definition Language (WSDL [22]). As web services are mostly generated from object-oriented (OO) programs, we can learn from the substantial research on OO metrics [1-3, 6, 7, 15, 16, 19, 21].

Hence, the objective of our study is to find indicative metrics for componentizing software components into Web services. By analyzing the WSDL artifacts of existing web services, this study first helps us to understand the nature of publicly available web services. Since WSDL documents have an interface definition without exposing implementation body, our study is mostly related to interface metrics [5, 11], however we found that such interface metrics for OO programs are not always useful to tell whether a software component is suitable to be made part of a Web service. Through quantitative analysis of the collected data from both Web services and software components, we have determined which software metrics are more useful to identify suitable candidates for Web service componentization.

The remainder of the paper is organized as follows. Section 2 explains how the data samples were obtained; Section 3 introduces interface metrics and uses the samples collected in Section 2 to compare the metrics of WSDL and other software component ; Section 4 does similar comparison with respect to the OO metrics; Section 5 discusses the related work. Section 6 discusses the threats to validity of our results and presents our conclusions and future work.

2. Data

In this section, we explain the method used in collecting the artifacts for the subsequent metrics-based analysis and comparison.

2.1 Web services

We collected descriptions of web services, i.e., WSDL documents, using three methods: (1) searching WSDL files using Google web service; (2) crawling on the Web for possible WSDL files starting from popular web service portals; and (3) collecting web service registered in UDDI servers.

Among the three methods, Google web service is the most effective in collecting WSDL files. We constructed a program to search for possible WSDL links by sending out queries automatically. When the results were larger than the 1000 limit imposed by Google, the program can partition the large result set to smaller ones by adding additional keywords. Hence our program retrieved most of the WSDL links provided by Google web service.

However, the results provided by Google web service and Google manual search are different. Although Google manual search indicates that there are around 30K possible web service links, Google web service only provides a small portion of the data (less than 10 %).

In order to find more WSDL documents, we also build a multithreaded web crawler to crawl automatically for WSDL links. The crawler starts with WSDL-rich web sites, such as *xmethods.com*, *webservicex.net*, *webservicelist.com*, *salcentral.com*, and *www.bindingpoint.com*. We crawled millions of links, and picked the ones that looked like WSDL URLs.

UDDI (Universal Description, Discovery and Integration) [18] registries that we searched include the one's provided by IBM and Microsoft. We build a program to exhaustively find all the web services on the servers by searching the names of the web services. Since registries have a limit on size of the returning result, we broke the large result set by iterating through more specific queries. Hence we retrieved all the items in the registries, both containing thousands of them. However, many of those registered items are not web services, or do not have a links to WSDL files. In the end, only 549 WSDL documents were collected by the UDDI method.

Having collected those WSDL documents from the three methods, we found out that there were many duplicates and syntactically incorrect WSDL files. After filtering out those files, we obtained 2710 syntactically correct and unique WSDL documents. We have provided a search and download interface for those WSDL documents at www.cs.uwindsor.ca/~jlu/wsdl

We should point out that this total number of WSDL files is much larger than several other WSDL collections reported in the literature so far. For example, Fan et al [10]'s study on a WSDL collection contains 640 WSDL documents, a WSDL search engine called Woogle [8] has 1213 WSDL documents in total summed from several subcategories, including possible duplicates; and *xmethods.com*, a popular WSDL portal, has only 515 WSDL documents.

2.2 Other software components

While selecting the software components counterparts to compare with web services, we consider the following criteria. First, the web services describe their software interfaces only. Hence the component to be compared with should have a clear distinction between interface and implementation. Second, WSDL use XML Schemas to describe the data structure and, in some sense, the application domain model. Since most software applications use an object model, we should compare the schema model in WSDL with the object model.

Many web services are wrapped from OO systems. In particular, XML Schemas in WSDL are mapped from classes of the underlying OO systems. The comparison with typical OO systems in terms of representative OO metrics would reveal the OO features in web services.

The OO systems to be compared with the web services are from [7], which also developed the popular OO metrics that is used in our paper. There are two libraries that are developed in different organizations. One library (called GUI) is developed by a software vendor for a GUI application, which consists of 634 classes written in *C++*. The other (called Manufacturing) is developed by a semiconductor manufacturer for flexible machine control and manufacturing system. It consists of 1459 classes written in *Smalltalk*. Unfortunately, the data source in [7] is not available. For improving the validity of our comparison, we decided also to measure large-scale open-source OO projects such that the observations can be replicated. Specifically, the Eclipse IDE release version 3.2.2 has been chosen (called Eclipse) for our comparison measurements. It consists of 89 plugin components, written in *Java*. The system has in total 15499 classes and about 2 million LOC.

Whilst a WSDL file does not describe the implementation details of the software component, it is straightforward to compare their interfaces with other interfaces. The other interfaces used in our comparison are from [5]. The data contains twelve components from various areas, including

- CoreAudio: a low level interface to Audio hardware written in C;
- DB: a high level interface for database querying, updating, and connection management;
- MFC: a large component which provides a C++ framework for Windows application.

3. Interface metrics

WSDL defines a set of operations, which is similar to interface definitions. Since there are no implementation details, we can not measure the metrics that are based on the complete source code, such as McCabe complexity [17] that measures the controls flow. What we can measure is based on the information in the interface only, such as operations, arguments, and identifiers. Following the interfaces metrics proposed in [5], we adopted those metrics for web services.

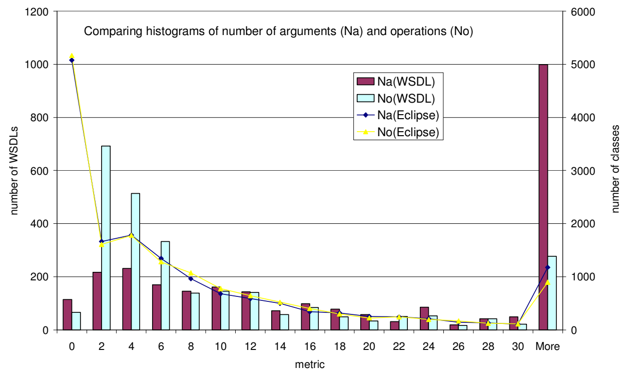
3.1 Arguments

Operation (n_o) and argument (n_a) counts provide a way to measure the size of web services. Isolated, the total number of operations and the total number of arguments are not very meaningful values. They tell us the functional size of the service. The smaller the number of operations and arguments, the better the understandability. On the other hand, the larger the number of operations and arguments, the better the chances for the Web service to be invoked and used. What really becomes interesting, is to look at the ratio between the number of arguments and operations [5]. One would expect that operations with fewer arguments will be easier to understand and to wrap as web services. In our study we use the APO metric defined below.

Definition 1(APO) *Given the total number of arguments* n_a *and total number of operations* n_o*, the argument per operation (APO) is defined as*

$$APO = \frac{n_a}{n_o} \qquad (1)$$

Argument count n_a is the total number of arguments in a WSDL document. Each *part* in a message will be counted as an argument. A response message can have multiple *parts*, i.e., more than one arguments. APO is similar to the interface metric *Argument Per Procedure* defined in [5].



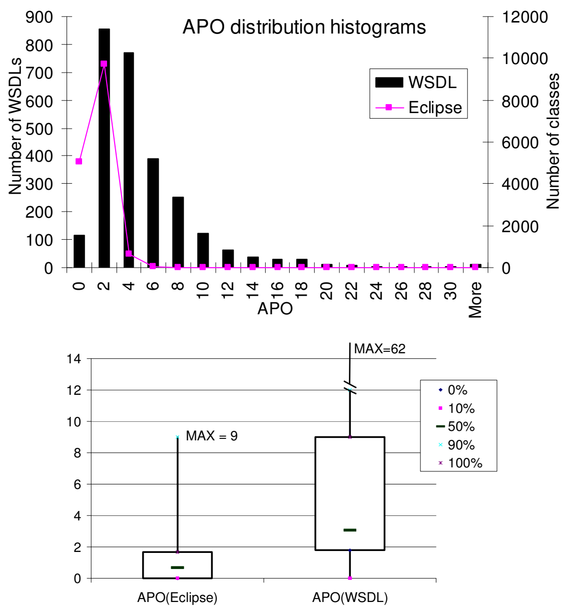


Figure 1 Comparing the APO metric. The histogram (upper charts) shows vertically the frequency of the metric horizontally distributed over ranges. The box plot (the lowest chart) contrasts the distributions to compare data points at the minimal (0%), maximal (100%), median (50%) and a box of percentile (10%~90%). For better visualization, here, the y-axis scale is such that the maximum APO for web services is beyond the visible area

Example 1 *(APO). Given the following WSDL document, there are two operations, i.e., checkPrice with one input and two output arguments, and checkAvailability with one input and one output arguments. Hence,* n_a *is 5 and* n_o *is 2. APO is 2.5.*

```
<?xml version="1.0"?>
<definitions name="PriceCheck" xmlns:pc="http://example/PriceCheck"
xmlns:avail="http://example/ns/availability"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap"
xmlns="http://schemas.xmlsoap.org/wsdl/">
<types>
 <xsd:schema>
      <xsd:element name="Sku" />
           <xsd:restriction base="xsd:string">
           <xsd:length value="8"/>
      </xsd:element>
  <xsd:complexType name="avail:priceType">
     <extension base="Sku"/>
   <xsd:sequence>
    <xsd:element name="shipping" type="xsd:double"/>
    <xsd:element name="price" type="xsd:double"/>
   </xsd:sequence>
  </xsd:complexType>
 </xsd:schema>
</types>
<message name="PriceCheckRequest">
  <part name="sku" type="Sku"/>
</message>
<message name="PriceCheckResponse">
 <part name="pcResult" type="avail:priceType"/>
</message>
<message name="AvailabilityCheckRequest">
  <part name="sku" type="Sku"/>
</message>
<message name="AvailabilityCheckResponse">
 <part name="amount" type="xsd:integer"/>
</message>
<portType name="PriceCheckPortType">
```

```
 <operation name="checkPrice">
  <input message="pc:PriceCheckRequest"/>
  <output message="pc:PriceCheckResponse"/>
 </operation>
<operation name="checkAvailability">
  <input message="pc:AvailabilityCheckRequest"/>
  <output message="pc:AvailabilityCheckResponse"/>
 </operation>
</portType>
<binding name="PriceCheckSOAPBinding"
        type="pc:PriceCheckPortType">
 <operation name="checkPrice"> … </operation>
 <operation name="checkAvailability"> … </operation>
</binding>
<service name="PriceCheckService">
 <port name="PriceCheck"
      binding="pc:PriceCheckSOAPBinding"/>
</service>
</definitions>
```

In Figure 1, we show a histogram of the APO metrics for the WSDL documents and contrast them with that of Eclipse components in a box plot. It is clear that with few exceptions, most WSDL documents have a larger number of APO.

Table 1 compares the APO of WSDL documents with other components. In OO programming, a class method saves one implicit argument (e.g., *this* pointer in C++, *self* object in Smalltalk and *this* reference in Java). As measured in [5], difference of OO and procedural programs can account for about one argument smaller in the APO. However, we observe that the average APO (WSDL) is much larger than APO (OO), which cannot be attributed only to the difference in OO and procedural languages.

Table 1: APO metric (median indicated when possible[1])

Component	Arguments	Operations	APO	Median APO
WSDL	199208	36187	5.50	3.07
CoreAudio[5]	42	134	3.19	N/A
DB [5]	210	404	1.92	N/A
MFC[5]	5907	7735	1.31	N/A
Eclipse	144106	136762	1.05	0.67

Observation 1 (Larger APO in WSDL due to stateless Web services) *WSDL are stateless, therefore their operations are designed to provide transaction-style services (similar to static class methods in OO), as opposed to stateful interface of objects. An object instance method often depends on the state of the instance variable (e.g.., **this** reference in Java) to provide additional information to the method parameters. In other words, without a persistency layer to store the temporary state into a database or into a file, WSDL operations must obtain all necessary information from the argument list. As a result, the APO metric will be much larger than those of object-oriented systems.*

Observation 2 (Larger APO in WSDL due to efficiency considerations to minimize the number of SOAP messages over the network) *In order to provide interoperability, the Web services messages must encode the parameters into an envelope of SOAP messages in the XML format, which implies larger message size than that in object-oriented programs. In addition, such SOAP messages are sent over the network in Web services, whereas the OO messaging can be simply a local invocation inside computer's memory. Taking efficiency into account, therefore, the WSDL arguments should encode as much information as possible in order to reduce the total number of messages sent across the network. As a result, the average number of parameters per request/response message is much larger.*

Most operations arguments in WSDL are defined by a request and a response message of certain XML schema type. These message types consist of parts in simple types such as *xsd:string* and *xsd:integer*.

Definition 2 (DAC, DAR) *Let the distinct argument count (DAC) be the cardinality of the set of (argument, type) pairs appeared in a WSDL interface description. DAR is defined as*

$$DAR = \frac{DAC}{n_a} \qquad (2)$$

Example 2 (DAR) *Given the WSDL description in Example 1, the set of the (argument, type) pairs is:* {(sku, Sku), (shipping, double), (price, double), (amount, integer)}. *Hence DAC=4, and DAR=4/5=0.8.*

Table 2 lists the DAR values for various components.

Table 2: Average DAR metrics

Libraries	DAC	DAR
WSDL	17	0.54
DB [5]	129	0.32
CoreAudio [5]	33	0.25
MFC [5]	2363	0.31

Observation 3 *WSDL documents have a high DAR, which means that most of them are developed independently with less similarities between arguments.*

Consider two WSDL documents both have three distinct arguments and twelve argument instances in total. Suppose in the first WSDL each distinct argument is repeated four times each. In the second WSDL, one distinct argument is used ten times, and two others are used once. The DAR values for both WSDLs would be the same. However, the second

[1] As most distributions of the metrics are skewed, i.e., non-symmetric, using the median provides a better estimator of the center of the distribution than the average.

interface is more consistent, and allows better data transferability. Hence ARS is introduced to capture this difference [7]

Definition 3 (ARS) *Given a sequence A of (name, type) pairs in the web service, the argument repetition scale (ARS) is:*

$$ARS(A) = \frac{\sum_{a \in A} |a|^2}{|A|} \qquad (3)$$

where |a| *denotes the number of repetition of a in A.*

Example 3 *For the example Web service, the (name, type) pair sequence is (sku, string), (sku, Sku), (shipping, double), (price, double), (amount, integer). Note that the pair (sku, Sku) appeared twice. Thus,*

$$ARS = \frac{1*2^2 + 3*1^2}{5} = \frac{4+3}{5} = 1.4$$

In OO programs, DAC is often large for complex systems. For example, DAC for Microsoft Foundation Classes (MFC) is 2363. On the other hand, WSDL documents do not exhibit large DAC. Table 3 compares the argument metrics of WSDL documents with that of OO components in [5].

Lower ARS indicates more specialized functionality and higher DAR indicates more consistency of argument declarations. Consistent argument declarations make it easier to understand and reuse the components. Having higher DAR and lower ARS, therefore, WSDL interfaces are more reusable in general, but also more cumbersome to understand the functionality of each operation.

Table 3: Average of ARS metrics

Libraries	DAC	ARS
WSDL	17	1.9
DB [5]	129	12.55
CoreAudio [5]	33	9.06
MFC [5]	2363	21.42

Observation 4 *Comparing with other components, WSDL tends to have higher DAR and smaller ARS, indicating that fewer arguments are repeated. This indicates that web services are coarse grained components with few operations that share the same parameters.*

3.2 Identifiers

Just as in programming languages, names used in WSDL should be self-explanatory and easy to understand. In the following, we measure the identifiers used in the WSDL documents.

Definition 4(Mean/Median Identifier Length, MIL/MeIL) *The mean identifier length (MIL) is the mean size (i.e. number of characters) of argument and operation names occurring in the WSDL documents. MeIL is the median length of identifiers*

Figure 2 shows the distribution of MIL in WSDL documents.

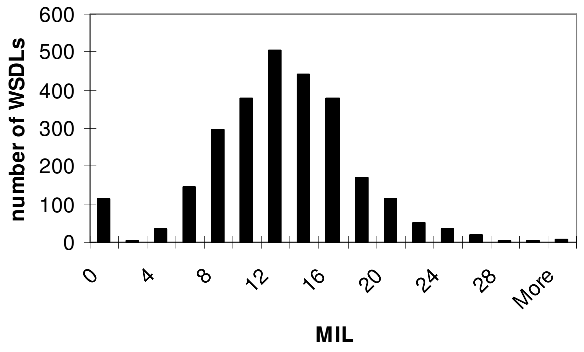


Figure 2: MIL metric histogram in web service

The mean string commonality (MSC) metric can be used to measure naming consistency. This can indicate the reusability of web services.

Definition 5 (MSC) *The Mean String Commonality of a set of identifiers A is defined as:*

$$MSC(A) = \frac{\sum_{(x,y) \in comb(A)} \frac{|lcs(x,y)|}{\max(|x|,|y|)}}{n*(n-1)/2} \qquad (4)$$

where n =|A|*, lcs(x, y) is the largest common substring of x and y, comb(A) is the set of all distinct combinations between the identifiers in A.*

Example 4 *In the given WSDL example, the set of argument names we measure for the interface is A={sku, shipping, price, amount}, MIL is 5.5, MeIL is 6. The set of operation names we measure for the interface is O={checkPrice, checkAvailability}. The MSC(A) and MSC(O) are calculated respectively as 0.181 and 0.588.*

Table 4 lists the MSC for arguments, denoted as MSC(A), and for operations, denoted as MSC(O) respectively for the WSDL documents and the published metrics in [5].

Table 4: Identifier metrics

Language	MIL	MeIL	MSC(A)	MSC(O)
WSDL	12.0	11	9.00	0.19
CoreAudio [5]	15.04	12	0.27	0.46
DB [5]	6.23	6	0.14	0.18
MFC [5]	7.69	7	0.17	0.21

Observation 5 *WSDL descriptions tend to have longer identifier length in general, except CoreAudio component. Both WSDL and CoreAudio describes interfaces specialized in small areas. Also, longer identifiers make WSDL more self-explanatory.*

Another reason for longer names may be due to the naming convention introduced by WSDL generation tools. Most of the names are generated automatically from conventional programs by adding some prefix or postfixes. For example, most message names end with 'response' or 'request'. For this reason, the MSC in WSDL arguments is much higher than that of other software components. The MSC for operations, however, is lower because the WSDL operations tend to be self-contained and have coarser granularity, therefore the naming does not share as much in common as other components.

4. OO Metrics

Operations in WSDLs use XML Schema to define their input and output data types. Since WSDL does not provide the implementation details, we can only measure metrics similar to WMC (Weighted Methods per Class), DIT (Depth of Inheritance Tree), and NOC (Number of Children) from the object-oriented metrics suite [7, 23]. Since many Schemas in WSDL are mapped from classes in other programming languages, it is natural to compare Schema structure with class structures using DIT and NOC.

WSDL size can be measured in terms of number of operations in the PortType. OPS (Operations Per Service) is similar to WMC (Weighted Methods per Class) in the OO metric suite.

Definition 6 (OPS) *OPS is the total count of operations that are declared by a PortType of a web service.*

Figure 3 shows the distribution of OPS in both WSDL and Eclipse. Compared with typical OO applications, the median OPS is similar to one OO system (GUI), and smaller than the others. The maximum OPS of WSDL is larger than all the OO systems.

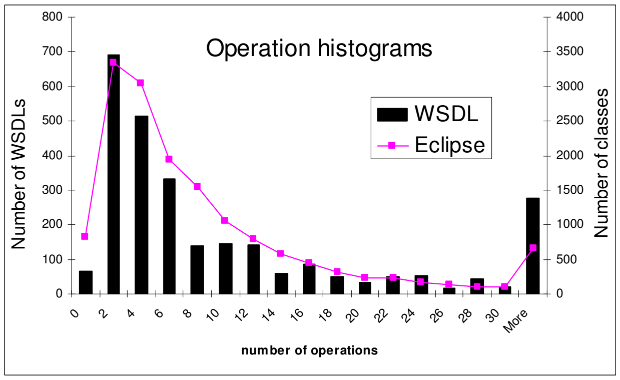


Figure 3 : Comparing OPS distribution in WSDL with the WMC distribution in Eclipse. Though histograms scale differently, they match in shape

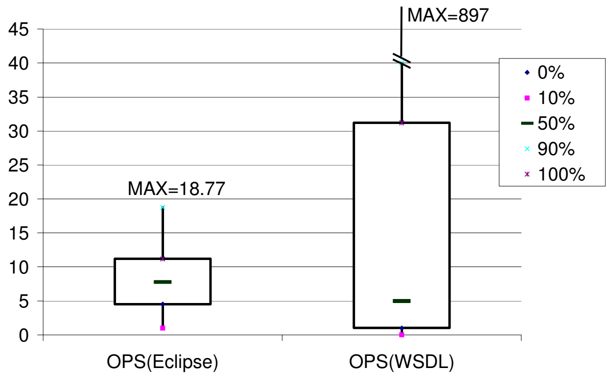


Figure 4: Comparing OPS metric using boxplot

One observation is that a few large web services have a huge number of operations. For example, 2.8MB size *eBay* has 120 operations and the *DotNetJunkies* service has 897 operations. On the one hand, it shows that web service design typically has larger number of services. Table 5 summarizes the OPS metric for the WSDL and the compared OO and Java components.

Table 5: Operation Per Service

	Median	Max	Min
WSDL	5	897	1
GUI (C++) [7]	5	106	1
Manufacturing (Smalltalk) [7]	10	346	1
C: Eclipse (Java)	8	711	1

Figure 3 shows the distribution of the OPS metric value among the Eclipse classes. Both distributions are very similar, which suggests that OPS is not a good metric to distinguish a software component from a web service interface. Figure 4 compares the OPS value in WSDL with Java components in box plot.

Example 5 *For the WSDL in Example 1, both XSD Schema extension and restriction have one child, hence NOC for those two types are 1.*

Figure 5 reveals a distribution of the NOC metric of WSDL documents and its comparison with Eclipse components. Most web services have very small number of direct children in the inheritance tree.

Definition 7(DIT) *DIT is defined as inheritance depth in XML Schema used in WSDL. .*

In a WSDL document, XML Schema definition is enclosed by the <types/> tags. Inside the XML Schema, inheritance relation is defined by the <extension base="baseType"/> or <restriction/> tag. Analyzing the inheritance relation of the types, we obtain a directed acyclic graph. DIT is the length of the longest inheritance path in the DAG.

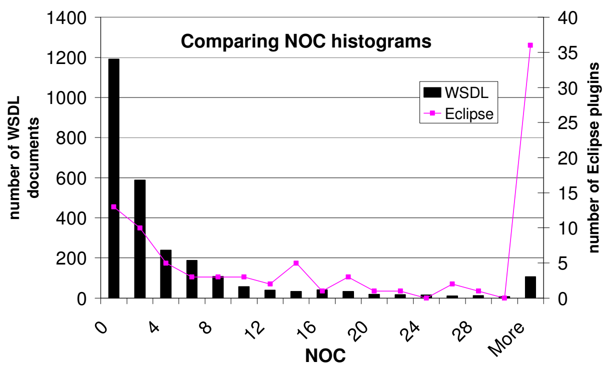


Figure 5: Comparing NOC metric distributions

Definition 8. (Number of Children) *NOC is the immediate number of children of a node in the XML Schema inheritance tree.*

Table 6 compares the median OO metrics of the components with that of the WSDL documents.

Table 6: Comparing the median/average OO metrics

Language	OPS/NOM	NOC	DIT
WSDL:Median Average	5.00 13.35	1.00 6.52	1.00 0.67
Eclipse: Median Average	7.78 8.82	5.62 71.26	5.00 5.24

Observation 6 *WSDL descriptions tend to have much larger spread in OPS than the other software WMC*[2] *because they are provided as an interface to a larger chunk of functionality: the service. On the other hand, the NOC and DIT of the web services are all smaller, indicating a limited reuse of the data structures revealed by the interfaces.*

Table 7 compares the DIT metric for WSDL documents with other software components.

Table 7: Comparison for the DIT metric

Language	Median	Max	Min
WSDL	1	5	0
GUI [7]	1	8	0
Manufacturing [7]	3	10	0
Eclipse	5	10	1

To have a better view of the difference of WSDL with other components in terms of DIT, we conducted a detailed comparison with Eclipse on the distribution. Figure 6 compares the distribution and boxplots of DIT of WSDL with that of Eclipse, which shows that the XML Schema in the WSDL typically have smaller DIT.

[2] We use *net.sourceforge.metrics* to collect metrics for Eclipse, where the WMC is weighted by the McCabe cyclomatic complexity. To make a fair comparison with WSDL interfaces for which it is impossible to obtain McCabe complexity, we use the number of methods per class/interface (NOM) instead.

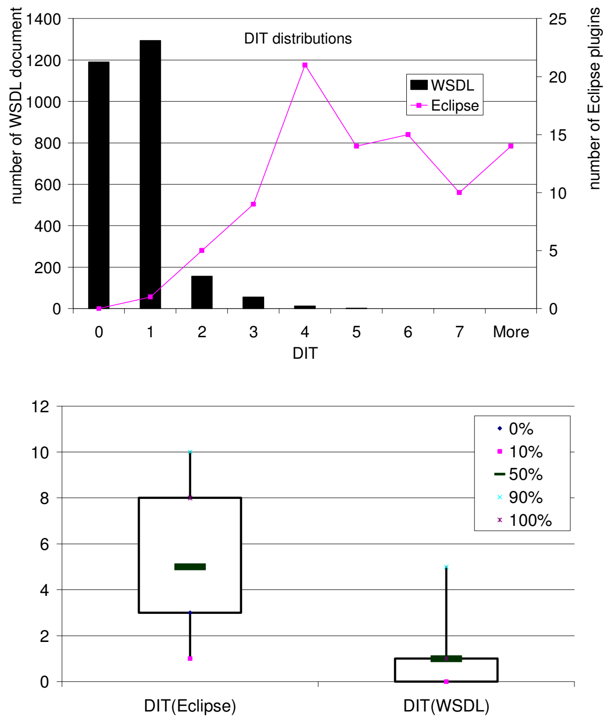


Figure 6: DIT metric comparison with Eclipse

5. Related work

Central to the Web Service technology is the Web Service Declaration Language (WSDL) that standardizes the interface among the loosely coupled Web Service components.

A few works [10, 14, 20] are done in collecting and analyzing public available web services. Fan et al [10] presents a snapshot of publicly available web services (640 in total) in the following three aspects: 1) application domains of web services, such as data source look up, banking, and sensing; 2) number of operation per service, and 3) documentation of web services, i.e., the length of documentation in each web service. The relevance of the survey was in connection with web service discovery and composition. For example, documentation in WSDL will help web service discovery using standard information retrieval techniques.

Kim et al [14] and Schmietendorf et al [20] survey features specific for web services, such as RPC style and document style of web services, liveness and delay, and SOAP message size. Kim et al [14] also analyze in detail the Amazon and Google web services. Schmietendorf et al [20] report techniques used to generate web services, frequency of changes of web services, styles of web service (RPC and document style). It also summarizes the number of methods in WSDLs and the number of arguments in each operation. Compared with these studies, we focus on the comparison with other traditional software components, in terms of existing software metrics.

Dong, X. et al [9] built a web service search and browsing engine, by clustering web services into different application domains. In a more general setting, our work is related to metrics for various software artifacts, including software applications, software components and interfaces, XML documents and XML Schemas, and OO programs.

6. Conclusions and future work

We compared web services with other software libraries in terms of interface, variable naming and OO metrics. The comparison reveals that operations in WSDL interfaces have more operands per operation than other software components (larger APO). The argument metrics comparison shows that web services share fewer common argument names and longer names are used in the arguments. Finally, the OO metrics reveals WSDL use less inheritance in the Schemas of WSDL description than typical OO software . Although these metrics are useful in differentiating typical existing WSDL from other software libraries, we also found that the number of operations per service is less useful in identifying suitable software components to be wrapped into web services.

Although we cannot guarantee the comparison is complete as it is practically impossible to collect all the existing WSDL, on one hand, and typical OO software, on the other to compare, with diversified samples, we believe they are quite representative to the existing web services and other software libraries. To minimize thread of validity, we conduct the comparison using open-source projects and well-established software metrics. The compared programming languages do not cover all languages in use, but we believe they are representative languages for existing software systems.

In the future, we are going to measure the argument metrics of the Eclipse components and further classify them using the interface metrics. It will be of particular interests to correlate these metrics with the quality, the popularity (e.g. number of hits), the domain type, and type of software development process of the Web Services and with the quality and other analogous characteristics of the OO systems. These metrics could also be used as part of a tool to identify in OO repositories candidates for wrapping into Web services. We also plan to formalize the results by doing some statistical testing (e.g. test whether the medians are similar or not at a given significant level).

Acknowledgement

We would like to thank Abu Hassan and Edyta Saklak for their help in building the programs to collect WSDL documents.

References

[1] J. Barnard, "A new reusability metric for object-oriented software," *Software Quality Control,* vol. 7, pp. 35-50, 1998.

[2] V. Basili, L. Briand, W. Melo, and lio, "A Validation of Object-Oriented Design Metrics as Quality Indicators," *IEEE Trans. Softw. Eng.,* vol. 22, pp. 751-761, 1996.

[3] J. Bieman, "Deriving Measures of Software Reuse in Object Oriented Systems," in *Proceedings of the BCS-FACS Workshop on Formal Aspects of Measurement*, 1992, pp. 63-83.

[4] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. F. Nielsen, S. Thatte, and D. Winer, "Simple Object Access Protocl (SOAP) 1.1."

[5] M. Boxall and S. Araban, "Interface Metrics for Reusability Analysis of Components," in *ASWEC '04: Proceedings of the 2004 Australian Software Engineering Conference (ASWEC'04)*, 2004.

[6] L. Briand, E. Arisholm, S. Counsell, F. Houdek, and T. e. Fosse, "Empirical Studies of Object-Oriented Artifacts, Methods, and Processes: State of the Art and Future Directions," *Empirical Software Engineering: An International Journal,* vol. 4, pp. 385-402, 1999.

[7] S. R. Chidamber and C. F. Kemerer, "A Metrics Suite for Object Oriented Design," *IEEE Trans. Softw. Eng.,* vol. 20, pp. 476-493, 1994.

[8] L. Ding, T. Finin, A. Joshi, R. Pan, R. S. Cost, Y. Peng, P. Reddivari, V. Doshi, and J. Sachs, "Swoogle: a search and metadata engine for the semantic web," ACM Press New York, NY, USA, 2004, pp. 652-659.

[9] X. Dong, A. Halevy, J. Madhavan, E. Nemes, and J. Zhang, "Similarity search for web services," in *VLDB Conference* Toronto, 2004.

[10] J. Fan and S. Kambhampati, "A snapshot of public web services," *SIGMOD Rec.,* vol. 34, pp. 24-32, 2005.

[11] M. Ghassemi and R. Mourant, "Evaluation of coupling in the context of Java interfaces (poster session)," in *OOPSLA '00 (Addendum)*, 2000, pp. 47-48.

[14] S. Kim and M. Rosu, "A survey of public web services," in *WWW Alt. '04: Proceedings of the 13th international World Wide Web conference on Alternate track papers \& posters*, 2004, pp. 312-313.

[15] L. Laranjeira, "Software Size Estimation of Object-Oriented Systems," *IEEE Trans. Softw. Eng.,* vol. 16, pp. 510-522, 1990.

[16] W. Li and S. Henry, "Maintenance metrics for the object oriented paradigm," in *Software Metrics Symposium, 1993. Proceedings., First International*, 1993, pp. 52-60.

[17] T. McCabe, "A Complexity Measure," *IEEE Trans. Software Eng.,* vol. 2, pp. 308-320, 1976.

[18] E. Newcomer, *Understanding Web Services: XML, WSDL, SOAP, and UDDI*: Addison-Wesley Professional, 2002.

[19] S. Purao and V. Vaishnavi, "Product metrics for object-oriented systems," *ACM Comput. Surv.,* vol. 35, pp. 191-221, 2003.

[20] A. Schmietendorf and R. R. Dumke, "Empirical analysis of available web services," in *Proc. of the 13th International Workshop on Software Measurement*, 2003, pp. 51-69.

[21] M.-H. Tang, M.-H. Kao, and M.-H. Chen, "An Empirical Study on Object-Oriented Metrics," in *METRICS '99: Proceedings of the 6th International Symposium on Software Metrics*, 1999.

[22] W3C, "Web Services Description Language (WSDL) 1.1," 2001.

[23] H. Washizaki, H. Yamamoto, and Y. Fukazawa, "A Metrics Suite for Measuring Reusability of Software Components," in *METRICS '03: Proceedings of the 9th International Symposium on Software Metrics*, 2003.