



# Revising WSDL Documents: Why and How, Part 2

In a previous article, the authors demonstrated that avoiding several common design errors in Web Service Description Language (WSDL) documents helps developers effectively discover Web services. Their proposed guidelines are unfortunately applicable only when publishers follow the top-down, or contract-first, method of building services, which lacks popularity due to its inherent costs. Here, they present an approach for preventing such errors when using a counterpart method — namely, bottom-up or code-first — and measure the approach's impact on service discovery.

**Cristian Mateos,  
Marco Crasso, and  
Alejandro Zunino**  
*Argentinian National Council for  
Scientific and Technical Research*

**José Luis Ordiales Coscia**  
*Universidad Nacional del Centro  
de la Provincia de Buenos Aires*

Service-oriented computing (SOC)<sup>1</sup> involves building new applications with reusable building blocks (services) that are described, discovered, and remotely consumed using standard protocols. The common technological choice for realizing SOC is Web services — programs with interfaces described by Web Services Description Language (WSDL) documents that developers can discover and consume via standard Web protocols. WSDL is based on XML and lets publishers describe their services' functionality as sets of abstract operations with inputs and outputs defined in the XML Schema Definition (XSD) language. They can also specify the associated binding information so clients can actually consume the offered operations. WSDL is commonly used to describe services that operate under RPC style. On the other hand, RESTful services follow the REST architectural style, which is more appropriate for services that map to the CRUD (create, read, update, delete) metaphor.

Developers describe RESTful services via the Web Application Description Language (WADL), which defines services as sets of resources that can be created, modified, and deleted. Several articles have compared WSDL- and WADL-based services, and information on the similarities and differences of both alternatives is available elsewhere.<sup>2</sup>

To effectively discover and understand Web services from their WSDL documents, publishers must pay special attention to the WSDL specification when developing services.<sup>3–5</sup> For contract-first Web services, in which the WSDL interface description for a service comes before its implementation, WSDL documents' understandability, legibility, and clarity (or taken together, *discoverability*)<sup>5</sup> improve significantly when we consider a catalog of common design bad practices, or *antipatterns*. However, the most popular approach for building Web services is code-first, which is based on first implementing a

Table 1. Core subset of Web service discoverability antipatterns.

| Antipattern                                   | Occurs when   |
|---|---|
| Ambiguous names                               | Ambiguous or meaningless names denote a WSDL document's main elements.  |
| Empty messages                                | Empty messages are used in operations that don't produce outputs or receive inputs.                                       |
| Enclosed data model                           | Data-type definitions used for exchanging information are placed in WSDL documents rather than in separate XSD documents. |
| Low cohesive operations in the same port type | Port types have weak semantic cohesion.   |
| Redundant data models                         | A WSDL document contains many data types for representing the same objects in the problem domain.                         |
| Whatever types                                | A special data type represents any object in the problem domain.  |

service and then automatically generating the corresponding WSDL document from the implemented code.<sup>6</sup> Developers do this using language-dependent tools such as Axis's Java2WSDL or the VisualStudio WSDL tool. The question that arises is whether developers can avoid antipatterns for code-first Web services early on, and if so, what impact this has on discoverability.

## Background

Standard-compliant approaches to Web service discovery are based on service descriptions specified in WSDL. Many discovery approaches extract keywords from WSDL documents and then model the extracted information as inverted indexes or vector spaces.<sup>7</sup> Then, developers use the generated models to retrieve relevant WSDL documents for a given keyword-based query. These approaches are strongly inspired by classic information retrieval techniques, such as word sense disambiguation, stop-word removal, and stemming. Despite their pragmatism, feasibility, and low adoption costs, their effectiveness is jeopardized by poorly written WSDL documents – that is, documents lacking proper comments, containing nonrepresentative, unrelated, or redundant keywords, and so on.<sup>3,4</sup>

Our previous study identifies recurrent bad practices occurring in a large dataset of public WSDL documents; measures their impact on service discovery in standard-compliant service registries and on users' perception; and proposes guidelines to remedy the identified problems as a catalog of WSDL antipatterns.<sup>5</sup> We classified these antipatterns with regard to which service

interface parts they entail, particularly how they use comments and identifiers, as well as how they model the data exchanged by services (see Table 1). The guidelines consist of refactoring actions that should be applied over the WSDL documents. Given a WSDL document with antipatterns, its publisher can methodically modify it until all antipatterns have been removed. However, these solutions are applicable only when following a contract-first approach.

Unfortunately, contract-first isn't popular among developers because it requires more effort than code-first. Publishers must master the WSDL specification and the XSD data-type language. On the other hand, code-first simply generates WSDL documents from existing service code. In the end, publishers focus on developing and maintaining service implementations in any programming language, while delegating WSDL generation to specialized tools during service deployment.<sup>6</sup> So, we need an approach for avoiding WSDL antipatterns when using the code-first method.

## Relationships between Implementations and Interfaces

With the code-first approach, relationships exist between a service implementation's source code and its associated WSDL document. We hypothesize that we can predict WSDL antipatterns by taking a set of metrics on service implementations. This rationale assumes that a typical code-first tool performs a mapping  $T : C \rightarrow W$ , where  $C = \{M(I_0; R_0); \dots; M_N(I_N; R_N)\}$  is the front-end class implementing a service.  $W = \{O_0(I_0; R_0); \dots; O_N(I_N; R_N)\}$  is the WSDL document that

describes the service containing a port type for the service implementation class and as many operations  $O$  as public methods  $M$  are defined in class  $C$ . Each operation of  $W$  is associated with one input message  $I$  and a return message  $R$ , while each message conveys an XSD type that models the corresponding class method's parameters. Code-first tools such as WSDL.exe, Java2WSDL, and gSOAP are based on a mapping  $T$  for generating WSDL documents from C#, Java, and C++, respectively. Each tool implements  $T$  in a particular manner, mostly owing to the targeted programming languages' different characteristics. So, we hypothesize that service implementations' measurable properties can influence the resulting WSDL documents.

We used an exploratory approach to test the statistical correlation among object-oriented (OO) metrics<sup>8</sup> and WSDL antipatterns. We aimed to evaluate the feasibility of avoiding WSDL antipatterns by considering OO metrics from code-implementing services. The idea was to use these metrics as sentinels that warn the user about potential antipattern occurrences early in a Web services implementation. Although several researchers have used OO metrics to predict the number of defects or other quality attributes in conventional software at development time,<sup>9</sup> our earlier approach associated implementation metrics with Web service interfaces.<sup>10</sup> The hypotheses in Table 2 present statistically significant relationships between the two variable types of a correlation model, given by OO metrics (independent variables) and antipatterns (dependent variables). Note that a correlation between two variables doesn't imply causation – that is, the former variable values aren't a sufficient condition for the latter ones. However, such a correlation means that the former variable values are a necessary condition for the latter ones. Although WSDL antipattern occurrences aren't strictly caused by OO metric values in their associated implementations, the mentioned WSDL antipatterns require certain OO metric values to be present in service implementations.

We used Spearman's rank correlation coefficient to model the relations. In the tests, we used a dataset of 154 real code-first services, which we collected via the Merobase component finder (<http://merobase.com>), the Exemplar engine (<http://tinyurl.com/7bytzxx>), and Google Code (<http://code.google.com>). Then, we collected OO metrics from service implementations using an

extended version of ckjm.<sup>11</sup> To measure the number of antipattern occurrences, we used an automatic WSDL antipattern-detection tool (<https://sites.google.com/site/easysoc/home/anti-patterns-detector>).<sup>12</sup> Table 3 shows the correlation between the OO metrics associated with the presented hypotheses and the antipatterns. Bolded values are those coefficients that are statistically significant at the 5 percent level – that is,  $p$ -value  $< 0.05$ . Decisions and validations underpinning this statistical analysis are available elsewhere.<sup>10</sup>

### Revising Your Service Implementations

The correlation among the *coupling between object classes* (CBO), *weighted method per class* (WMC), *abstract type count* (ATC), and *empty parameters method* (EPM) metrics and the antipatterns, which were statistically significant for the analyzed Web service dataset, suggests that, in practice, an increment/decrement of the metric values taken on implementing a code-first Web service can affect antipattern occurrences in the service's generated WSDL document. To confirm this, we conducted five rounds of refactoring that in turn produced five new datasets – one for each metric and a fifth in which we included all the previous refactorings. Thus, *metric-driven refactorings* are those that, when applied to the source code, result in variations on the WMC, CBO, ATC, and EPM metric values.

The first metric-driven refactoring we considered was Move Method refactoring,<sup>13</sup> which split original services containing more than one operation into two new services so that, on average, WMC in the refactored services represented 50 percent of its original value. This refactoring resulted in a new dataset,  $DS_{WMC}$ , that contained approximately twice as many services as the original. Next, we focused on CBO by using the inverse of the Infer Generic Type Arguments<sup>13</sup> refactoring to replace every occurrence of a complex data type with the Java primitive type `String` to create a dataset,  $DS_{CBO}$ . In a third refactoring round, we focused on the ATC metric by replacing generic arguments (declared as `Object`, or `Collection` if not parametrized) with concrete ones, using Introduce Type Parameter refactoring. This produced the dataset  $DS_{ATC}$ . The last metric we considered was EPM. The refactoring we applied in this case introduced a new Boolean parameter to those methods without parameters. Again, we employed Introduce Type

Table 2. Hypotheses.

| Hypothesis   | Description   |
|--|---|
| $H_1 : CBO \rightarrow \text{Enclosed data model}$                           | The more classes directly related to the class implementing a service (CBO* metric <sup>8</sup> ), the more the <i>enclosed data model</i> antipattern occurs. CBO counts how many methods or instance variables defined by other classes a given class accesses. In the resulting WSDL documents, code-first tools typically include as many XSD definitions as the number of objects exchanged by service class methods. Increasing the number of external objects that service classes access can increase the likelihood of data-type definitions within WSDL documents.                            |
| $H_2 : WMC \rightarrow \text{Low cohesive operations in the same port type}$ | The more public methods belonging to the class implementing a service (WMC metric <sup>8</sup> ), the more the <i>low cohesive operations in the same port type</i> antipattern occurs. WMC counts a class's methods. More methods increase the probability that any pair are unrelated — that is, have weak cohesion. Because code-first tools map each method onto an operation, a higher WMC might increase the possibility that resulting WSDL documents have low cohesive operations.  |
| $H_3 : WMC \rightarrow \text{Redundant data models}$                         | The more public methods belonging to the class implementing a service (WMC metric <sup>8</sup> ), the more the <i>redundant data models</i> antipattern occurs. The number of message elements defined within a WSDL document is equal to the number of operation elements multiplied by two. Because each message can be associated with a data type, the likelihood of redundant data-type definitions increases with the number of public methods; this, in turn, increases the number of operation elements.  |
| $H_4 : WMC \rightarrow \text{Ambiguous names}$                               | The more public methods belonging to the class implementing a service (WMC metric), the more the <i>ambiguous names</i> antipattern occurs. Similar to $H_3$ , an increment in the number of methods can lift the number of nonrepresentative names within a WSDL document, given that for each method, a code-first tool automatically generates in principle five names (one for the operation, two for the I/O messages, and two for the data types).  |
| $H_5 : ATC \rightarrow \text{Whatever types}$                                | The more method parameters belonging to the class implementing a service that are declared nonconcrete data types (ATC metric), the more the <i>whatever types</i> antipattern occurs. Our ATC metric computes the number of method parameters that don't use concrete data types, or that use Java generics with type variables instantiated with nonconcrete data types. Code-first tools usually map abstract data types and badly defined generics onto the <code>xsd:any</code> constructor, which has been identified as the root cause for the <i>whatever types</i> antipattern. <sup>3,5</sup> |
| $H_6 : EPM \rightarrow \text{Empty messages}$                                | The more public methods belonging to the class implementing a service that don't receive input parameters (EPM metric), the more the <i>empty messages</i> antipattern occurs. We designed the EPM metric to count the number of methods in a class that don't receive parameters. Increasing the number of methods without parameters can increase the likelihood of empty messages antipattern occurrences, because code-first tools map this kind of method onto an operation associated with one input message element not conveying XSD definitions.   |

CBO: coupling between object classes; WMC: weighted method per class; ATC: abstract type count; EPM: empty parameters method.

Parameter refactoring. This generated the dataset  $DS_{EPM}$ . We performed a final round of refactoring by deriving a new dataset,  $DS_{ALL}$ , that included the other four refactorings.

Table 4 shows the average antipattern occurrences per WSDL document before (“Original” column) and after (“ $DS_x$ ” column) the refactoring

process. To generate the WSDL documents, we used Axis's Java2WSDL, the most popular Java tool for building code-first Web services. From the results, we can see that decreasing the OO metric values produced the same effect on their associated antipatterns as the one Table 3 shows. Concretely, reducing the WMC value by 50 percent

**Table 3. Most significant correlations between object-oriented metrics and antipatterns.**

| Antipattern/OO metric                         | WMC                   | CBO                   | ATC                   | EPM                   |
|---|-----------------------|-----------------------|-----------------------|-----------------------|
| Enclosed data model                           | 0.41                  | <b>0.98</b> ( $H_1$ ) | 0.12                  | 0.16                  |
| Low cohesive operations in the same port type | <b>0.61</b> ( $H_2$ ) | 0.38                  | 0.12                  | 0.39                  |
| Redundant data models                         | <b>0.79</b> ( $H_3$ ) | 0.33                  | 0.15                  | 0.31                  |
| Ambiguous names                               | <b>0.86</b> ( $H_4$ ) | 0.42                  | 0.25                  | 0.33                  |
| Whatever types                                | 0.50                  | 0.35                  | <b>0.60</b> ( $H_5$ ) | 0.32                  |
| Empty messages                                | 0.54                  | 0.20                  | 0.19                  | <b>0.99</b> ( $H_6$ ) |

**Table 4. Refactoring impact on antipatterns for Java2WSDL.**

| Metric and antipatterns                       | Original (average) | $DS_{WMC}$ (average) | $DS_{CBO}$ (average) | $DS_{ATC}$ (average) | $DS_{EPM}$ (average) | $DS_{ALL}$ (average) |
|---|--------------------|----------------------|----------------------|----------------------|----------------------|----------------------|
| Ambiguous names                               | 20.02              | 10.79                | 20.02                | 20.02                | 20.96                | 11.24                |
| Low cohesive operations in the same port type | 24.62              | 8.19                 | 24.62                | 24.62                | 19.04                | 6.25                 |
| Enclosed data model                           | 3.28               | 2.61                 | 0.04                 | 3.25                 | 3.28                 | 0.01                 |
| Whatever types                                | 0.83               | 0.43                 | 0.62                 | 0.00                 | 0.83                 | 0.00                 |
| Redundant data models                         | 52.96              | 15.10                | 132.96               | 53.89                | 57.81                | 34.10                |
| Empty messages                                | 0.94               | 0.44                 | 0.94                 | 0.94                 | 0.00                 | 0.00                 |
| Total average number of antipatterns          | 102.66             | 37.58                | 179.21               | 102.72               | 101.92               | 51.59                |

reduced the ambiguous names, low cohesive operations in the same port type, and redundant data model antipatterns by 53.89, 33.26, and 28.51 percent, respectively. We obtained similar results when refactoring for the CBO, ATC, and EPM metrics, producing an average reduction of the enclosed data model, whatever types, and empty messages antipatterns of 1.21, 100.00, and 100.00 percent, respectively.

We also observe that, although the individual metric-driven refactorings positively affected their associated antipatterns, some also increased the number of occurrences of other antipatterns. For example, the CBO metric caused a decrease in enclosed data model antipattern occurrences but also a considerable increase in the redundant data models antipattern. Furthermore, this increment's negative impact outweighs the benefits of the refactoring because the total number of antipatterns is higher with respect to the original dataset. This is a clear trade-off in which the service developer must analyze and select among different metric-driven service implementation alternatives. Two other metrics represent trade-offs. By

decreasing the ATC metric, resulting WSDL documents present fewer occurrences of the whatever types antipattern than the original WSDL document. However, this increases redundant data models. A similar situation occurs with the EPM metric and the empty messages and redundant data models antipatterns. Refactoring for the WMC metric is safe, in the sense that it doesn't present trade-offs, and by modifying its value, no undesired collateral effects appear.

Finally, note that when we applied all the refactorings to the same dataset ( $DS_{ALL}$ ), we reduced the total number of antipatterns with respect to the original dataset, but it was slightly higher than the one we obtained by applying only WMC refactoring. Considering that code refactoring is a time-consuming process, we can conclude that if the goal is to minimize the total number of antipatterns, the most efficient choice when refactoring is to focus only on WMC.

## Discovering Revised Code-First WSDL Documents

We performed another experiment to assess the impact our statistical-based approach for removing



antipatterns had on WSDL documents' discoverability. Methodologically, the evaluation consisted of three steps. First, we divided the set of code-first WSDL documents into two groups. One group consisted of those documents that were generated after we applied each proposed refactoring to the service implementations gathered from several open source projects. The other group consisted of the original versions of the WSDL documents. We refer to this latter group as the original. We next supplied two service registries with both groups of WSDL documents. Then, we queried the employed registries using one query per available service operation in the original group. For each query, we analyzed the position at which either the original WSDL document containing the operation or its refactored counterpart – known as precision-at- $n$  – were retrieved. Precision-at- $n$  computes precision at different cut-off points. For example, if the top five documents are all relevant to a query, and the next five are all nonrelevant, we have a precision of 100 percent at a cut-off of five documents, but a precision of 50 percent at a cut-off of 10 documents. Finally, we averaged the results over the total number of queries.

For the experimentation, we used a publicly available registry implementation of our previous service discovery approach (<https://sites.google.com/site/easysoc/home/service-registry>)<sup>14</sup> and Lucene4WSDL,<sup>5</sup> a modified version of Lucene (<http://lucene.apache.org>). For a given keyword-based query, these standard-compliant approaches return an ordered list of candidate WSDL documents, sorted by their similarity to the query.

For fairness, we built the employed queries from the original service implementations' source code – that is, the original dataset. We assumed that if developers want to replace an operation with a functionally equivalent one from an external service, they will probably use the name of the replaced operation as a query. This assumption is analogous to the query-by-example concept.<sup>14</sup> For example, the query for looking for operations functionally equivalent to one whose signature is `getActiveWorkflows(userID:string)` might be “get active workflows.” In fact, the employed registries split combined words within queries. Following this assumption, we obtained 879 queries, one per offered operation. Finally, we associated two WSDL documents with each query, one document belonging to the original group and another from

the refactored one. For the association, we selected the WSDL documents containing the operation needed. So, an important part of the evaluation involved determining from which positions in the registries candidate list the pair of original-refactored documents was retrieved.

We calculated the precision-at- $n$  results for each query with  $N$  in  $[1,10]$  – that is, the actual number of relevant services up to only  $N$  candidates in the result list. As in our previous experiment,<sup>5</sup> we chose this window size because we wanted a good balance between the number of candidates and the number of relevant candidates retrieved; a developer can easily examine up to 10 Web service descriptions.

Figure 1 shows the precision-at- $n$  results for the five original dataset combinations. We averaged the results for the 879 queries and smoothed them using Bézier curves. Additionally, each part of the figure shows the results when we used both Web Services Query By Example (WSQBE) and Lucene4WSDL via two colored pairs of curves. The continuous curves represent the WSQBE results, whereas the dashed curves illustrate Lucene4WSDL results. Brown and blue curves represent the original and refactored WSDL documents, respectively.

Graphically, we can quickly see from Figure 1 that blue curves begin at a higher point than brown ones for all the subfigures. This means that precision-at-1 was higher for the refactored WSDL documents using both registries. Moreover, this trend continues until the precision-at-4 results, where, for all subfigures, the original and refactored curves overlap. As various experiments demonstrate, better precision-at-1 and precision-at-2 have a significant impact on discoverability because users tend to select the highest-ranked search results first.<sup>15</sup> For instance, the probability that a user accesses the first ranked result is 90 percent, whereas the probability for accessing the next one is, at most, 60 percent.<sup>15</sup> So, our results empirically show that refactored WSDL documents are more discoverable than original ones.

**W**eb services descriptions are crucial for enabling truly global SOC. Such descriptions, in particular WSDL documents, are built by following either the contract-first or the code-first method. Our previous article<sup>5</sup> provides contract-first followers with guidelines for avoiding WSDL antipatterns (which we report on elsewhere<sup>6</sup>) in

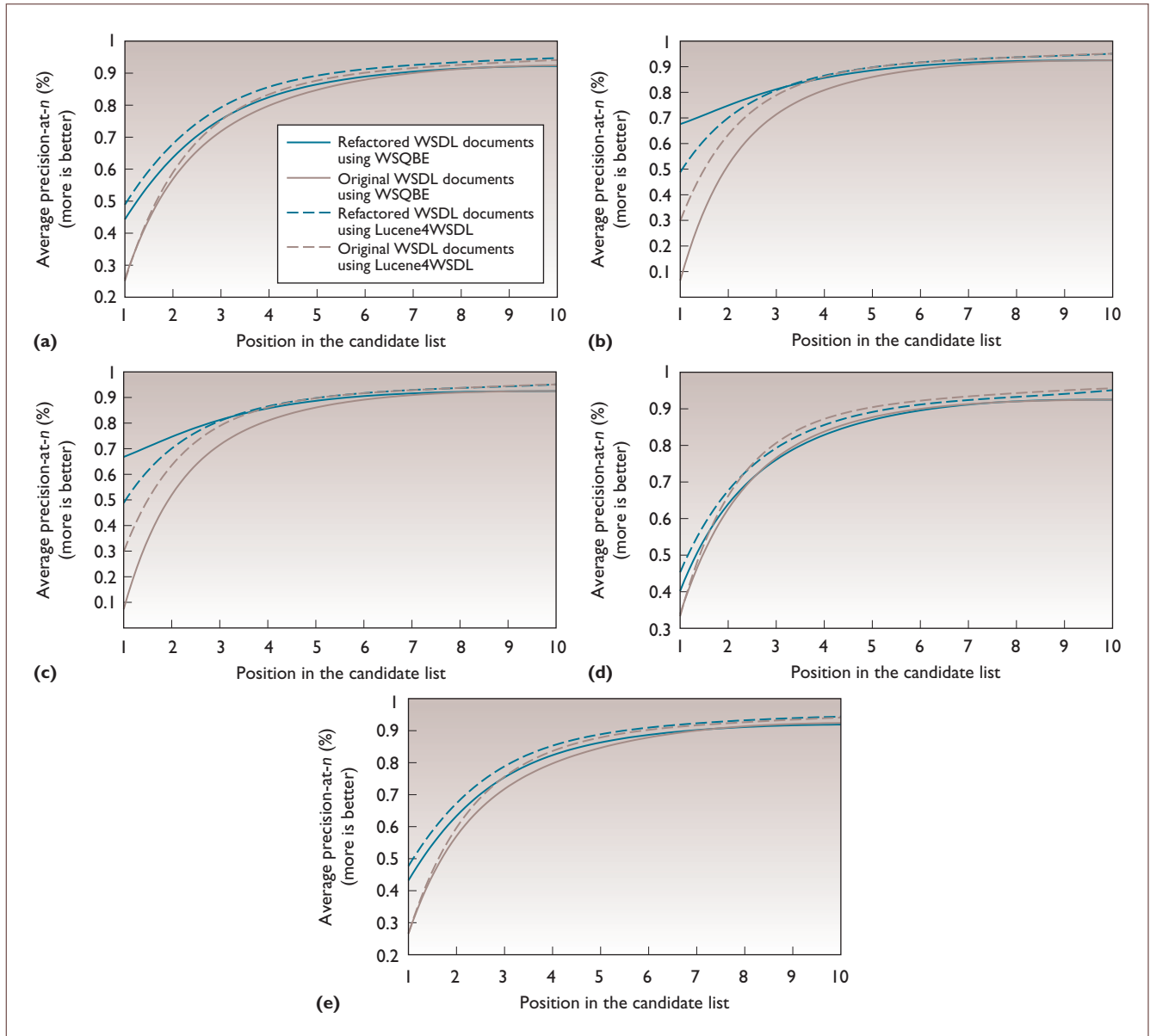



Figure 1. Averaged precision-at-n results comparison. We can see results for the following datasets: (a)  $DS_{WMC}$ ; (b)  $DS_{CBO}$ ; (c)  $DS_{ATC}$ ; (d)  $DS_{EPM}$ ; and (e)  $DS_{ALL}$ .

their WSDL documents. This article complements the previous two by presenting statistical evidence showing that we can eliminate antipatterns in code-first WSDL documents simply by looking at classic OO code metrics. Moreover, we've empirically demonstrated the feasibility of metric-driven refactorings for improving service discoverability. These contributions represent a starting point for addressing several open research questions, such as determining the relationship between service implementation metrics and other WSDL-based metric suites. Because empirical evidence shows

that by refactoring code-first Web services early, the resulting WSDL documents are more discoverable than their original counterparts, this article contributes to building better services in terms of discoverability.

Our study's main limitation is that the tool used for mapping from service implementations onto WSDL documents might have influenced the correlation between service implementation and service interfaces metrics, and we haven't yet measured this factor. Changing the code-first tool isn't always a smooth process, and we

must emphasize preparing each project of the dataset for each selected tool's specific requirements. For example, to use the WSPProvide tool, a developer must properly annotate a service implementation. This is the main reason why we didn't measure the influence of code-first tools in this study, although we plan to do so in the future. Recall, however, that Java2WSDL is the most popular code-first tool. 

## Acknowledgments

We acknowledge the financial support provided by the Agencia Nacional de Promoción Científica y Tecnológica through grant PAE-PICT 2007-02311.

## References

1. M.N. Huhns and M.P. Singh, "Service-Oriented Computing: Key Concepts and Principles," *IEEE Internet Computing*, vol. 9, no. 1, 2005, pp. 75–81.
2. P. Adamczyk et al., "Rest and Web Services: In Theory and in Practice," *REST: From Research to Practice*, Springer, 2011, pp. 35–57.
3. J. Pasley, "Avoid XML Schema Wildcards for Web Service Interfaces," *IEEE Internet Computing*, vol. 10, no. 3, 2006, pp. 72–79.
4. M.B. Blake and M.F. Nowlan, "Taming Web Services from the Wild," *IEEE Internet Computing*, vol. 12, no. 5, 2008, pp. 62–69.
5. M. Crasso et al., "Revising WSDL documents: Why and How," *IEEE Internet Computing*, vol. 14, no. 5, 2010, pp. 48–56.
6. J.M. Rodriguez et al., "Bottom-Up and Top-Down Cobol System Migration to Web Services: An Experience Report," *IEEE Internet Computing*, vol. 17, no. 2, 2013, pp. 44–51.
7. M. Crasso, A. Zunino, and M. Campo, "A Survey of Approaches to Web Service Discovery in Service-Oriented Architectures," *J. Database Management*, vol. 22, no. 1, 2011, pp. 103–134.
8. S. Chidamber and C. Kemerer, "A Metrics Suite for Object Oriented Design," *IEEE Trans. Software Eng.*, vol. 20, no. 6, 1994, pp. 476–493.
9. T. Gyimothy, R. Ferenc, and I. Siket, "Empirical Validation of Object-Oriented Metrics on Open Source Software for Fault Prediction," *IEEE Trans. Software Eng.*, vol. 31, no. 10, 2005, pp. 897–910.
10. C. Mateos et al., "Detecting WSDL Bad Practices in Code-First Web Services," *Int'l J. Web and Grid Services*, vol. 7, no. 4, 2011, pp. 357–387.
11. D. Spinellis, "Tool Writing: A Forgotten Art?" *IEEE Software*, vol. 22, no. 4, 2005, pp. 9–11.
12. J.M. Rodriguez, M. Crasso, and A. Zunino, "An Approach for Web Service Discoverability Antipatterns Detection," *J. Web Eng.*, vol. 12, nos. 1 & 2, 2013, pp. 131–158.
13. M. Fowler et al., *Refactoring: Improving the Design of Existing Code*, Addison-Wesley Professional, 1999.
14. M. Crasso, A. Zunino, and M. Campo, "Combining Query-by-Example and Query Expansion for Simplifying Web Service Discovery," *Information Systems Frontiers*, vol. 13, no. 3, 2011, pp. 407–428.
15. E. Agichtein et al., "Learning User Interaction Models for Predicting Web Search Result Preferences," *Proc. 29th Ann. Int'l ACM SIGIR Conf. Research and Development in Information Retrieval*, ACM, 2006, pp. 3–10.

**Cristian Mateos** is a full-time teaching assistant at the Universidad Nacional del Centro de la Provincia de Buenos Aires (UNICEN) and a member of the Instituto de Sistemas Tandil (ISISTAN) and the Argentinian National Council for Scientific and Technical Research (CONICET). His research interests include parallel and distributed programming, grid middleware, and service-oriented computing. Mateos has a PhD in computer science from UNICEN. Contact him at [cmateos2006@gmail.com](mailto:cmateos2006@gmail.com); [www.exa.unicen.edu.ar/~cmateos](http://www.exa.unicen.edu.ar/~cmateos).

**Marco Crasso** was a researcher at the Argentinian National Council for Scientific and Technical Research (CONICET) and a faculty member of the Instituto de Sistemas Tandil (ISISTAN) at the Universidad Nacional del Centro de la Provincia de Buenos Aires (UNICEN) until April 2013. His research interests include Web service discovery and programming models for service-oriented architectures. Crasso has a PhD in computer science from UNICEN. Contact him at [mcrasso@gmail.com](mailto:mcrasso@gmail.com); [www.exa.unicen.edu.ar/~mcrasso](http://www.exa.unicen.edu.ar/~mcrasso).

**Alejandro Zunino** is an adjunct professor at Universidad Nacional del Centro de la Provincia de Buenos Aires (UNICEN) and a member of Instituto de Sistemas Tandil (ISISTAN) and the Argentinian National Council for Scientific and Technical Research (CONICET). His research areas include grid computing, service-oriented computing, Semantic Web services, and mobile agents. Zunino has a PhD in computer science from UNICEN. Contact him at [alejandro.zunino@isistan.unicen.edu.ar](mailto:alejandro.zunino@isistan.unicen.edu.ar); [www.exa.unicen.edu.ar/~azunino](http://www.exa.unicen.edu.ar/~azunino).

**José Luis Ordiales Coscia** has an MSc from the Universidad Nacional del Centro de la Provincia de Buenos Aires (UNICEN). His thesis is about early detection of WSDL bad practices in code-first services. Ordiales Coscia has a BSc in systems engineering from UNICEN. Contact him at [jlordiales@gmail.com](mailto:jlordiales@gmail.com).