Revising WSDL documents: Why and How - Part II

Cristian Mateos* Marco Crasso* Alejandro Zunino*
José Luis Ordiales Coscia**

February 4, 2013

Abstract

In a previous paper [1], we have shown that effectively discovering Web services is subject to avoiding a number of common design errors in publishers' Web Service Description Language (WSDL) documents. We have therefore proposed guidelines, which unfortunately are applicable only when publishers follow the top-down, a.k.a. contract-first, method to build services, which is not very popular due to its inherent costs. We present an approach to prevent such errors when using its counterpart method, namely bottom-up or code-first, and measure the impact of the approach in service discovery. The rationale behind the study is that since code-first services interfaces are automatically generated by tools that given a service implementation deterministically map programming languages constructs onto WSDL elements, the measurable properties of services implementations may influence resulting services interfaces.

Keywords: Web Services Modeling, Services Architectures, Web Services Description Language, Web services publishing, Services discovery process and methodology

1 Introduction

Service-Oriented Computing (SOC) [2] refers to reusable building blocks (*services*) that are described, discovered and remotely consumed by using standard protocols to build new applications. The common technological choice for materializing SOC is Web Services, which are programs with interfaces described by WSDL documents that can be discovered and consumed via standard Web protocols. Particularly, WSDL bases on XML and allows publishers to describe the functionality of their services as a set of abstract operations with inputs and outputs defined in the XML Schema Definition (XSD) language, and to specify the associated binding information so that clients can actually consume the offered operations. The WSDL is commonly used for describing services that operate under the RPC style. On the other hand, RESTful services follow the Representational State Transfer (REST) architectural style, which is more appropriate for services that map to the CRUD metaphor. RESTful services are described by means of the WADL (Web Application Description Language), a language to describe services as set of resources that can be created, modified, and deleted. The comparison of WSDL and WADL based services has been the focus of several articles from which [3] is a good one for understanding the similarities and differences of both alternatives.

It has been shown that, for the process of discovering and understanding Web Services from their WSDL documents to be effective, publishers must pay special attention to WSDL specification upon developing services [4, 5, 1]. For the case of *contract-first* Web Services, in which the WSDL interface description for a service comes before its implementation, taking into account a catalog of common design bad practices –i.e. anti-patterns– significantly improves understandability, legibility and clarity (from now on discoverability) of WSDL documents [1]. However, the most popular approach to build Web Services is *code-first*, which bases on first implementing a service and then automatically generating the corresponding WSDL from the implemented code [6]. This is done by using language-dependent tools such as Axis' Java2WSDL, or VisualStudio WSDL tool. The question that arises is then whether it is possible to early avoid anti-patterns for code-first Web Services or not, and what the impact on discoverability is.

*ISISTAN Research Institute - Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET) and **UNICEN University. Campus Universitario, Tandil (B7001BBO), Buenos Aires, Argentina. Tel.: +54 (249) 4439682 ext. 35. Fax.: +54 (249) 4439683

2 Background

Standard-compliant approaches to Web Service discovery are those based on services descriptions specified in WSDL. There are many discovery approaches that extract keywords from WSDL documents, and then model the extracted information as inverted indexes or vector spaces [7]. Then, the generated models are used for retrieving 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-words removal, and stemming. Despite their pragmatism, feasibility, and low cost of adoption, their effectiveness is jeopardized by poorly written WSDL documents, i.e., those lacking proper comments, containing non-representative, unrelated or redundant keywords, and so on [5, 4].

The study presented in [1] identifies recurrent bad practices that take place in a large data-set of public WSDL documents, measures their impact on service discovery in standard-compliant service registries and users' perception, and proposes guidelines to remedy the identified problems as a catalog of WSDL anti-patterns. The authors classified the anti-patterns as regarding with how services interfaces are described, particularly how comments and identifiers are employed, and how the data exchanged by services are modeled (see Table 1). The guidelines consist of refactoring actions that should be applied over the WSDL documents. Then, given a WSDL document having anti-patterns, its publisher can methodically modify it until all anti-patterns have been removed. However, these solutions can be applied when following contract-first only.

Table 1: The core sub-set of the Web Service discoverability anti-patterns

Anti-pattern	Occurs when
Ambiguous names	Ambiguous or meaningless names are used for denoting the main elements of a WSDL document.
Empty messages	Empty messages are used in operations that do not produce outputs nor 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	Many data-types for representing the same objects of the problem domain coexist in a WSDL document.
Whatever types	A special data-type is used for representing any object of the problem domain.

However, contract-first is not 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 WSDLs from existing service code. In the end, publishers focus on developing and maintaining services implementations in any programming language, while delegating WSDL generation to specialized tools during service deployment [6]. Therefore, an approach to avoid WSDL anti-patterns when using the code-first method is needed.

3 Relationships between service implementations and interfaces

When using code-first, there are relationships between the source code of a service implementation and its associated WSDL document. We set forth the hypothesis that WSDL anti-patterns may be predicted by taking a set of metrics on services implementations. This rationale assumes that a typical code-first tool performs a mapping $T : C \rightarrow W$, being $C = \{M(I_0, R_0), .., M_N(I_N, R_N)\}$ the frontend class implementing a service, and $W = \{O_0(I_0, R_0), .., O_N(I_N, R_N)\}$ the WSDL document describing 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. Moreover, 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 parameters of the corresponding class method. 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, though each tool implements T in a particular manner mostly because of the different characteristics of the targeted programming languages. Therefore we set the hypothesis that the measurable properties of services implementations may influence resulting WSDL documents.

We used an exploratory approach to test the statistical correlation among OO metrics and the WSDL anti-patterns. The goal is to evaluate the feasibility of avoiding WSDL anti-patterns by taking into account Object-Oriented (OO) metrics from the code implementing services. The idea is employing these metrics as sentinels that warn the user about the potential occurrence of anti-patterns early in the implementation of Web Services. Although several researchers have been using OO metrics to predict the number of defects or other quality attributes in conventional software at development time [9], with respect to Web Services, the approach to associate implementation metrics with services interfaces has been recently and exclusively explored in [10]. We found that the hypotheses of Table 2 present statistical significant relationships between the two kinds of variables of a correlation model, given by OO metrics (independent variables) and anti-patterns (dependent variables). It is worth noting that a correlation between two variables does not imply causation, i.e., the former variable values are not a sufficient condition for the latter variable ones. However, such a correlation means that the former variable values are a necessary condition for the latter ones. Therefore, WSDL anti-patterns occurrences are not strictly caused by OO metrics values in their associated implementations, but the mentioned WSDL anti-patterns require certain OO metrics values to be present in services implementations.

We used the Spearman's rank correlation coefficient to model the relations. In the tests, we used a data-set of 154 different real code-first services, which were 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 by using an extended version of *ckjm* [11]. For measuring the number of anti-pattern occurrences, we employed an automatic WSDL anti-pattern detection tool [12] (`https://sites.google.com/site/easysoc/home/anti-patterns-detector`). Table 3 shows the correlation between the OO metrics associated with the hypotheses presented and the anti-patterns. The values in bold are those coefficients which are statistically significant at the 5% level, i.e., p-value < 0.05, which is a common choice when performing statistical studies. For space reasons, all decisions and validations underpinning this statistical analysis can be found in [10].

4 Revising Your Services Implementations

The correlation among the WMC, CBO, ATC and EPM metrics and the anti-patterns, which were found to be statistically significant for the analyzed Web Service data-set suggests that, in practice, an increment/-decrement of the metric values taken on the implementation of a code-first Web Service may affect anti-patterns occurrences in its generated WSDL. To confirm this, we conducted five rounds of refactoring which in turn produced five new data-sets, one for each of the four mentioned metrics, and a fifth one where all the previous refactorings were included. Thus, we refer as "metric-driven refactorings" to those refactorings that, when applied to the source code, result in variations on the values of the WMC, CBO, ATC and EPM metrics.

The first metric-driven refactoring considered was the Move Method refactoring [13], to split original services that contained more than one operation into two new services so that on average WMC in the refactored services represented a 50% of its original value. This refactoring resulted in a new data-set DS_{WMC} that contained approximately twice as many services as the original one. Next, we focused on CBO by using the inverse of the Infer Generic Type Arguments [13] refactoring to replace every occurrence of a complex data-type with the Java primitive type String to create a data-set DS_{CBO}. In a third refactoring round, we focused on the ATC metric, by replacing generic arguments (i.e., declared as Object, or Collection not being parametrized) with concrete ones, using the Introduce Type Parameter refactoring. This produced the data-set DS_{ATC}. The last metric considered was EPM. The refactoring applied in

Table 2: Hypotheses

Hypothesis	Description
$H_1 : CBO \rightarrow$ *Enclosed data model*	The higher the number of classes directly related to the class implementing a service (CBO metric [8]), the more the *Enclosed data model* anti-pattern occurrences. CBO counts how many methods or instance variables defined by other classes are accessed by a given class. Code-first tools typically include in resulting WSDL documents as many XSD definitions as objects are exchanged by the methods of service classes. Increasing the number of external objects that are accessed by service classes may increase the likelihood of data-type definitions within WSDL documents.
$H_2 : WMC \rightarrow$ *Low cohesive operations in the same port − type*	The higher the number of public methods belonging to the class implementing a service (WMC [8] metric), the more the *Low cohesive operations in the same port-type* anti-pattern occurrences. WMC counts the methods of a class. A greater number of methods increases the probability that any pair of them are unrelated, i.e., having weak cohesion. Since code-first tools map each method onto an operation, a higher WMC may increase the possibility that resulting WSDL documents have low cohesive operations.
$H_3 : WMC \rightarrow$ *Redundant data models*	The higher the number of public methods belonging to the class implementing a service (WMC metric [8]), the more the *Redundant data model*s anti-pattern occurrences. The number of *message* elements defined within a WSDL document is equal to the number of *operation* elements multiplied by two. As each *message* may be associated with a data-type, we believe that the likelihood of redundant data-type definitions increases with the number of public methods, since this in turn increases the number of *operation* elements.
$H_4 : WMC \rightarrow$ *Ambiguous names*	The higher the number of public methods belonging to the class implementing a service (WMC metric), the more the *Ambiguous names* anti-pattern occurrences. Similarly to H_3, an increment in the number of methods may lift the number of non-representative names within a WSDL document, since for each method a code-first tool automatically generates in principle five names (one for the operation, two for the input/output messages, and two for the data-types).
$H_5 : ATC \rightarrow$ *Whatever types*	The higher the number of method parameters belonging to the class implementing a service that are declared as non-concrete data-types (ATC metric), the more the *Whatever types* anti-pattern occurrences. ATC (Abstract Type Count) is a metric of our own that computes the number of method parameters that do not use concrete data-types, or use Java generics with type variables instantiated with non-concrete data-types. Code-first tools usually map abstract data-types and badly defined generics onto the xsd:any constructor, which has been identified as the root cause for *the Whatever types* anti-pattern [4, 1].
$H_6 : EPM \rightarrow$ *Empty messages*	The higher the number of public methods belonging to the class implementing a service that do not receive input parameters (EPM metric), the more the *Empty messages* anti-pattern occurrences. We designed the EPM (Empty Parameters Methods) metric to count the number of methods in a class that do not receive parameters. Increasing the number of methods without parameters may increase the likelihood of the *Empty messages* anti-pattern occurrences, because code-first tools map this kind of methods onto an operation associated with one input *message* element not conveying XSD definitions.

Table 3: Most significant correlations between OO metrics and anti-patterns

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

this case was to introduce a new boolean parameter to those methods not having parameters. Again we employed the INTRODUCE TYPE PARAMETER refactoring. This generated the data-set DS_{EPM}. Finally, a last round of refactoring was performed by deriving a new data-set, called DS_{ALL}, that included the other four refactorings.

Table 4: Refactoring: impact on anti-patterns for Java2WSDL

Metric and anti-patterns	**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 anti-patterns	102.66	37.58	179.21	102.72	101.92	51.59

Table 4 shows the average anti-patterns occurrences per WSDL document before (see the Original column) and after (see the DS_x column) the refactoring process. To generate the WSDL documents, we used Axis' Java2WSDL, the most popular Java tool for building code-first Web Services. From the results it can be seen that decreasing the values of the OO metrics produced the same effect on their associated anti-patterns shown in Table 3. Concretely, reducing the value of WMC by 50% reduced the *Ambiguous names*, *Low cohesive operations in the same port-type* and *Redundant data models* anti-patterns by 53.89%, 33.26% and 28.51%, respectively. Similar results were obtained when refactoring for the CBO, ATC and EPM metrics, producing an average reduction of the *Enclosed data model*, *Whatever types* and *Empty messages* anti-patterns of 1.21%, 100.00% and 100.00%, respectively.

It can also be observed that, while the individual metric-driven refactorings had a positive impact on their associated anti-patterns, some of them also increased the number of occurrences of other anti-patterns. For example, the CBO metric caused a decrease of the *Enclosed data model* anti-pattern occurrences but also a considerable increase on the *Redundant data models* anti-pattern. Furthermore, the negative impact of this increment outweighs the benefits of the refactoring since the total number of anti-patterns is higher with respect to the original data-set. This is a clear trade-off in which the service developer should analyze and select among different metric-driven service implementation alternatives. Two other metrics represent trade-offs. For example, by decreasing the ATC metric, resulting WSDL documents present less

occurrences of the *Whatever types* anti-pattern 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* anti-patterns. Refactoring for the WMC metric is safe, in the sense that it does not present trade-offs and by modifying its value no undesired collateral effects appear.

Finally, it is worth noting that when all the refactorings were applied on the same data-set (DS_{ALL}) the total number of anti-patterns were reduced with respect to the original data-set, but it was slightly higher than the one obtained by applying only the 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 anti-patterns, the most efficient choice when refactoring is to focus only on WMC.

5 Discovering Revised Code-first WSDL Documents

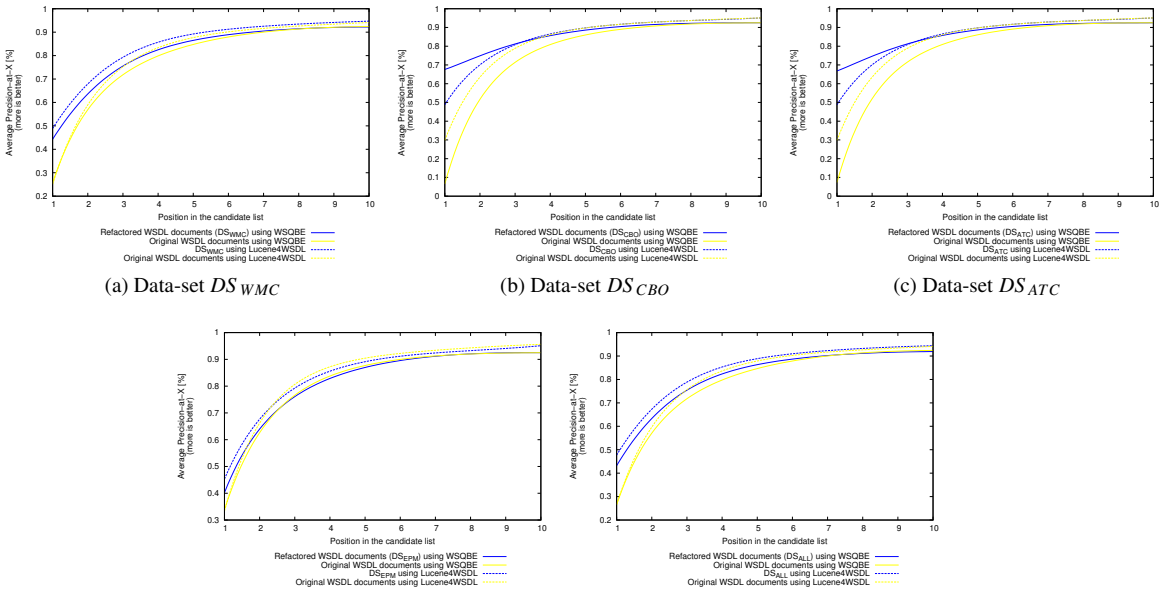
We performed another experiment to assess the impact on WSDLs discoverability of the explained statistical-based approach to remove anti-patterns. Methodologically, the evaluation consisted of three steps. First, the set of code-first WSDL documents was divided in two groups. One group consisted of those WSDL documents that were generated after applying each proposed refactorings to the service implementations gathered from several open source projects. Another group had the original versions of the WSDL documents. We refer to this latter group as "Original". Second, we supplied two service registries with both groups of WSDL documents. Third, we queried the employed registries using one query per available service operation in the original group. For each query we analyzed in which position were retrieved either the original WSDL document containing the operation or its refactored counterpart, which is formally known as Precision-at-*n*. Precision-at-*n* computes precision at different cut-off points. For example, if the top 5 documents are all relevant to a query and the next 5 are all non-relevant, we have a precision of 100% at a cut-off of 5 documents but a precision of 50% at a cut-off of 10 documents. Finally, we averaged the results over the total number of queries.

For the experimentation, a publicly available registry implementation of the approach to service discovery presented in [14] (`https://sites.google.com/site/easysoc/home/service-registry`) and Lucene4WSDL [1], a modified version of Lucene (`http://lucene.apache.org/`), were employed. For a given keyword-based query, these standard-compliant approaches to Web Service discovery return an ordered list of candidate WSDL documents, sorted according to how similar to the query they are.

For the sake of fairness we built the employed queries from the source code of the original service implementations, i.e. the Original data-set. This is because we assumed that if developers want to replace an operation with a functionally equivalent operation that is provided by 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 presented in [14]. For example, the query for looking for operations functionally equivalent to an operation whose signature is "getActiveWorkflows(userID:string)" may be "get active workflows". In fact, the employed registries split combined words within queries. Following this assumption, 879 queries were built, 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. Therefore, an important part of the evaluation consisted on checking in which positions of the registries candidate list the pair of original-refactored documents were retrieved.

The Precision-at-*n* results have been calculated for each query with N in [1,10], i.e., the actual number of relevant services up to only N candidates in the result list. Like the experiment presented in [1], we have chosen this window size because we want a good balance between the number of candidates and the number of relevant candidates retrieved, and a developer can easily examine up to 10 Web Service descriptions.

Figure 1 shows the Precision-at-*n* results for the five Original-DS_x combinations. The Precision-at-*n* results have been averaged for the 879 queries, and smoothed using Bézier curves. Additionally, each sub-figure shows the results when employing both registries, namely WSQBE and Lucene4WSDL. To do this, each sub-figure shows two colored pair of curves. The continuous curves represent the WSQBE results, while the dashed curves stand for Lucene4WSDL results. Finally, yellow and blue curves represent the original and refactored WSDL documents, respectively.



(a) Data-set DS_{WMC} (b) Data-set DS_{CBO} (c) Data-set DS_{ATC}

(d) Data-set DS_{EPM} (e) Data-set DS_{ALL}

Figure 1: Averaged Precision-at-n results comparison.

Graphically, it can be quickly observed from Figure 1 that blue curves start higher than yellow ones for all the sub-figures. 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, in which for all sub-figures the original and refactored curves overlap. As supported by different experiments, better Precision-at-*1* and Precision-at-*2* have a great impact on discoverability because users tend to select the highest ranked search results first [15]. For instance, the probability that a user accesses the first ranked result is 90%, whereas the probability for accessing the next one is, at most, 60% [15]. Therefore, our results empirically show that refactored WSDL documents are more discoverable than original ones.

6 Concluding remarks

Web Services descriptions are crucial for enabling truly global Service-Oriented Computing. Such descriptions, in particular WSDL documents, are built by following either the contract-first or the code-first method. The 14^{th} volume of IEEE Internet Computing enlightens contract-first followers with guidelines to avoid the WSDL anti-patterns reported in [6] on their WSDL documents [1]. The main contributions of this article, as a complement, are the presentation of statistical evidence showing that code-first WSDL documents can get rid of the same anti-patterns, by simply looking at classic OO code metrics, and the empirically demonstrated feasibility of metric-driven refactorings to improve services discoverability. These contributions represent a starting point for addressing several still open research questions, such as what is the relationship between services implementations metrics and other WSDL-based metric suites. Since empirical evidence shows that by early refactoring code-first Web Services, the resulting WSDL documents are more discoverable than their original counterparts, this paper contributes to build better services, in terms of discoverability.

The main limitation of the present study is that the tool employed for mapping from services implementations onto WSDL documents might have influenced the correlation between services implementations metrics and services interfaces ones, and we have not measured this factor yet. Changing the code-first tool is not always a smooth process, and we have to place effort on preparing each project of the data-set for the specific requirements of each selected tool. For example, for using WSProvide, a developer must place proper annotations on a service implementation. This is the main reason why code-first tools influence was

not measured in this study, although this is something we plan to do in the future. Recall however that Java2WSDL is the most popular code-first tool.

Acknowledgments

We acknowledge the financial support provided by ANPCyT through grant PAE-PICT 2007-02311.

References

[1] M. Crasso, J. M. Rodriguez, A. Zunino, and M. Campo, "Revising WSDL documents: Why and How," *IEEE Internet Computing*, vol. 14, no. 5, pp. 48–56, 2010.

[2] M. N. Huhns and M. P. Singh, "Service-oriented computing: Key concepts and principles," *IEEE Internet Computing*, vol. 9, no. 1, pp. 75–81, 2005.

[3] P. Adamczyk, P. Smith, R. Johnson, and M. Hafiz, "Rest and web services: In theory and in practice," in *REST: From Research to Practice*, pp. 35–57, Springer New York, 2011.

[4] J. Pasley, "Avoid XML schema wildcards for Web Service interfaces," *IEEE Internet Computing*, vol. 10, pp. 72–79, 2006.

[5] M. B. Blake and M. F. Nowlan, "Taming Web Services from the wild," *IEEE Internet Computing*, vol. 12, no. 5, pp. 62–69, 2008.

[6] J. M. Rodriguez, M. Crasso, C. Mateos, A. Zunino, and M. Campo, "Bottom-up and top-down cobol system migration to web services: An experience report," *IEEE Internet Computing*, 2011. in press.

[7] M. Crasso, A. Zunino, and M. Campo, "A survey of approaches to Web Service discovery in Service-Oriented Architectures," *Journal of Database Management*, vol. 22, no. 1, pp. 103–134, 2011.

[8] S. Chidamber and C. Kemerer, "A metrics suite for object oriented design," *IEEE Transactions on Software Engineering*, vol. 20, no. 6, pp. 476–493, 1994.

[9] T. Gyimothy, R. Ferenc, and I. Siket, "Empirical validation of Object-Oriented metrics on open source software for fault prediction," *IEEE Transactions on Software Engineering*, vol. 31, no. 10, pp. 897–910, 2005.

[10] C. Mateos, M. Crasso, A. Zunino, and J. L. Ordiales Coscia, "Detecting WSDL bad practices in code-first Web Services," *International Journal of Web and Grid Services*, vol. 7, pp. 357–387, 2011.

[11] D. Spinellis, "Tool writing: A forgotten art?," *IEEE Software*, vol. 22, pp. 9–11, 2005.

[12] J. M. Rodriguez, M. Crasso, and A. Zunino, "An approach for Web Service discoverability anti-patterns detection," *Journal of Web Engineering*, vol. in Press, 2012.

[13] M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts, *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Professional, 1 ed., July 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, pp. 407–428, 2011.

[15] E. Agichtein, E. Brill, S. Dumais, and R. Ragno, "Learning user interaction models for predicting web search result preferences," in *29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 3–10, 2006.

Cristian Mateos (http://www.exa.unicen.edu.ar/~cmateos) received a Ph.D. degree in Computer Science from the UNICEN, in 2008, and his M.Sc. in Systems Engineering in 2005. He is a full time Teacher Assistant at the UNICEN and member of the ISISTAN and the CONICET. He is interested in parallel/distributed programming, Grid middlewares and Service-oriented computing. Contact him at cmateos@conicet.gov.ar.

Marco Crasso (http://www.exa.unicen.edu.ar/~mcrasso) received a Ph.D. degree in Computer Science from the UNICEN in 2010. He is a member of the ISISTAN and the CONICET. His research interests include Web Service discovery and programming models for SOA. Contact him at mcrasso@conicet.gov.ar.

Alejandro Zunino (http://www.exa.unicen.edu.ar/~azunino) received a Ph.D. degree in Computer Science from the UNICEN, in 2003, and his M.Sc. in Systems Engineering in 2000. He is a full Adjunct Professor at UNICEN and member of the ISISTAN and the CONICET. His research areas are Grid computing, Service-oriented computing, Semantic Web Services and mobile agents. Contact him at azunino@conicet.gov.ar.

José Luis Ordiales Coscia received a BSc. degree in Systems Engineering from the UNICEN in 2011. Currently, he is an M.Sc. candidate working under the supervision of Cristian Mateos and Marco Crasso. His thesis is about early detection of WSDL bad practices in code-first services. Contact him at jlordiales@gmail.com.