



## SOAP and RESTful web service anti-patterns: A scoping review

Fuad Alshraideh<sup>1,2</sup>, Norliza Katuk<sup>2</sup>

<sup>1</sup>Ministry of Education, Jordan, fuad.alshraideh@yahoo.com

<sup>2</sup>School of Computing, Universiti Utara Malaysia, Malaysia, k.norliza@uum.edu.my

### ABSTRACT

Web services provide a uniform framework to achieve a high level of interaction between distributed heterogeneous software systems and data resources shared over the Internet. Producing a well-designed web service is significant because it leads to a more understandable service and a higher level of interaction and leads to effective software maintainability. However, web service is suffering from a poor design problem named anti-patterns. Analysis of the literature returned a plethora of studies on anti-patterns that caused difficulties for developers to synthesize and summarized the possible types of anti-patterns and further comprehend each of them. Due to this limitation, this paper aims to provide organized literature on the types of anti-patterns found in web services. A scoping review was conducted by searching scholarly documents, analyzing, and classified them based on their anti-pattern types. The review provided in this paper could be used as a guide for developers to identify the anti-patterns that could be found in web services.

**Key words:** Anti-pattern, Anti-pattern Detection, Web Service, Web Service Design, Interface Design

### 1. INTRODUCTION

Web service is a group of loosely coupled applications, self-describing software, that can be published, located and accessed flexibly and smoothly [1]-[3]. It provides a uniform framework to achieve a high level of interaction between the distributed heterogeneous software system and data resource sharing over the Internet [4]-[6]. In a simple word, web service can be described as independent software components that accept requests and return responses over the Internet. Web services are the components used in establishing service-oriented architecture (SOA) architecture [7]-[12] that converts the Internet from a repository of data to a repository of interactive services [13],[14]. Web service technology also opens a new cost-efficient form for software engineers to quickly develop and publish web applications by dynamically combining their applications with other published web service components to execute new business transactions [15].

There are two major protocols of web service implementation; first, Simple Object Access Protocol (SOAP) and second, Representational State Transfer (RESTful). SOAP is the earliest web service protocol that communicating data in Extensible Markup Language (XML) format and transported them in various messaging protocols primarily Hypertext Transfer Protocol (HTTP), Simple Mail Transfer Protocol (SMTP), and File Transfer Protocol (FTP). On the other hand, RESTful is a newer protocol that uses HTTP for transporting data in a various format including XML, Hypertext Markup Language (HTML), JavaScript Object Notation (JSON) and plaintext.

Web service can be openly accessed by external web applications regardless of the platform in which it is developed or deployed [3]. It is possible as it is invoked through its machine-processable format, which called web service interface [16],[17]; a set of operations that depicts interactions between service requesters and web service functionality [18]. Web services are programmed in similar way programmers programmed other computer-based systems. Like other computer-based systems, anti-patterns problems are found in web services due to inappropriate programming practices, designs, and implementations [19]. It caused accessibility issues in which the function of the services is difficult to comprehend [20],[21]; thus, making the process of designing interface complex and low quality. Eventually, the web service becomes abandoned due to low usage [22].

Many researchers attempt to improve anti-patterns problem in web services through detecting the anti-patterns and recommend the solutions to overcome the problem. However, web service anti-pattern problem is complex to resolve completely as each type of anti-pattern requires different approaches and distinct from others. Therefore, many past studies in the literature have been studying a particular type of web service anti-pattern. The increase of independent studies in anti-patterns has caused the need for synthesizing the available literature, summarizing and disseminating the types of anti-patterns so that it could help developers to avoid such anti-patterns; hence, producing high-quality and maintainable web services. As far as the authors are aware, no study synthesizing the types of web-service anti-patterns that exist. Apart from that, a question arises whether similar anti-patterns happen in the two

prominent web service protocols (i.e., SOAP and RESTful); or they just totally distinct. Therefore, this paper intends to address the limitation in the current studies by reviewing the literature on the types of web service anti-pattern for SOAP and RESTful, and further find overlapping problems.

The paper has three sections. Section 2 discusses the method for conducting the study. Next, Section 3 describes the results of the study based on specific research questions. Then, section 4 summarizes the findings.

## 2. METHOD

A scoping review [23]-[26] was conducted to identify the possible anti-patterns found in RESTful and SOAP web services. They are five stages of the process in conducting the scoping review as illustrated in Figure 1. The process of the scoping review starts with identifying research questions. Then an electronic search was conducted to identify relevant studies and followed by selecting the studies for further review. After that, the data are recorded and tabulated, and finally, the results were synthesized, summarized, and reported in this paper. A scoping review is a kind of review that provides a systematic way of collecting background information [27]. It is also an effective method to focus on the relevant literature to the researchers aiming at providing a quick mapping of the main concepts that underpin the research [28].

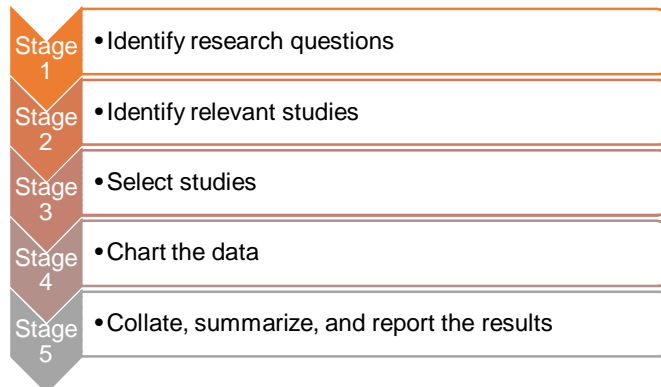


Figure 1: The process for conducting the scoping review [25],[26]

### Stage 1: Identify research questions

This scoping review aims to synthesize and summarize the possible types of web-service anti-patterns. As there are two prominent web service protocols (i.e., SOAP and RESTful); the number of anti-patterns could be substantial. Therefore, the authors analyzed the anti-patterns based on their occurrence found on the protocol style they were developed. Three research questions guide the scoping review as below:

RQ1: What are the types of anti-pattern found in SOAP web service?

RQ2: What are the types of anti-pattern found in RESTful web service?

RQ3: What are the common anti-patterns of SOAP and RESTful?

### Stage 2: Identify relevant studies

A comprehensive electronic document search on Google Scholar was done using the search phrase “web service anti-patterns”. It focused on documents written in English and published between the year 2000 and September 2019. The search results returned 673 documents, as summarized in Table 1. Year-based searching was conducted to get accurate search results. The search returned no documents for the year 1999. It could be justified by the fact that the web service technology has not emerged yet.

### Stage 3: Select studies

A filtering process was conducted to select the relevant documents that provide information on the types of anti-patterns. The study selected scholarly articles, including journal articles, conference articles, theses, and technical reports. Books and presentation slides were excluded from the results. Finally, fifty-two documents were selected, and the full list of the documents are listed in Table 2. The earliest study on web service anti-patterns was found published in 2016. The number of the study gradually and consistently growing; however, at a slow pace.

### Stage 4: Chart the data

The bibliographic information, abstract, and full-text of the documents were acquired and maintained using EndNote reference manager. The individual full-text documents were further analyzed to obtain the types of web-service anti-patterns that the researchers studied. The analysis was conducted based on the pre-defined RQs.

### Stage 5: Collate, summarize, and report the results

Finally, the analysis were systematically recorded in word processing documents, and they were further synthesized and summarized as reported in Section 3 of this paper.

Table 1: The number of total documents and relevant documents selected for the scoping study

Year	Total Documents Found	Number of relevant documents
2000	1	0
2001	1	0
2002	1	0
2003	7	0
2004	7	0
2005	7	0
2006	17	2
2007	17	1
2008	24	2
2009	41	1
2010	46	4
2011	46	3
2012	70	1
2013	55	4
2014	54	3
2015	49	6
2016	59	4
2017	69	6
2018	66	8
Sept 2019	36	7
<b>TOTAL</b>	<b>673</b>	<b>52</b>

**Table 2:** List of the documents included in this study.

Num.	Year	Authors	Title
1	2006	Kokash [29]	“A Comparison of Web Service Interface Similarity Measures”
2	2006	Zheng and Krause [30]	“Asynchronous Semantics and Anti-Patterns for Interacting Web Services”
3	2007	He and Yen [31]	“Adaptive User Interface Generation for Web Services”
4	2008	Beaton, et al. [32]	“Usability Challenges for Enterprise Service-Oriented Architecture APIs”
5	2008	Hacigümüs [33]	“Anti-Patterns: Integrating Distributed and Heterogeneous Data Sources in SOAs”
6	2009	Král and Žemlicka [34]	“Popular SOA Antipatterns”
7	2010	Crasso, et al. [35]	“Revising WSDL Documents: Why and How”
8	2010	Rodriguez, et al. [36]	“Improving Web Service Descriptions for Effective Service Discovery”
9	2010	Rodriguez, et al. [37]	“The EASYSOC Project: A Rich Catalog of Best Practices for Developing Web Service Applications”
10	2010	Rodriguez, et al. [38]	“Automatically Detecting Opportunities for Web Service Descriptions Improvement”
11	2011	Batra and Bawa [39]	“Semantic Discovery of Web Services Using Principal Component Analysis”
12	2011	Mateos, et al. [40]	“Detecting WSDL Bad Practices in Code-First Web Services”
13	2011	Rodriguez, et al. [41]	“Bottom-Up and Top-Down COBOL System Migration to Web Services”
14	2012	Mateos, et al. [42]	“Avoiding WSDL Bad Practices in Code-First Web Services”
15	2013	Coscia, et al. [43]	“Anti-Pattern Free Code-First Web Services for State-Of-The-Art Java WSDL Generation Tools”
16	2013	Han, et al. [44]	“Definition and Detection of Control-Flow Anti-Patterns in Process Models”
17	2013	Palma, et al. [45]	“SOA Antipatterns: An Approach for their Specification and Detection”
18	2013	Rodriguez, et al. [46]	“An Approach for Web Service Discoverability Anti-Pattern Detection for Journal of Web Engineering”
19	2014	Coscia, et al. [47]	“Refactoring Code-First Web Services for Early Avoiding WSDL Anti-Patterns: Approach and Comprehensive Assessment”
20	2014	Palma, et al. [48]	“Detection of REST Patterns and Antipatterns: A Heuristics-Based Approach”
21	2014	Torkamani and Bagheri [49]	“A Systematic Method for Identification of Anti-Patterns in Service-Oriented System Development”
22	2015	Mateos, et al. [50]	“Measuring the Impact of The Approach to Migration in the Quality of Web Service Interfaces”
23	2015	Mateos, et al. [51]	“A Tool to Improve Code-First Web Services Discoverability Through Text Mining Techniques”
24	2015	Mateos, et al. [52]	“A Stitch in Time Saves Nine: Early Improving Code-First Web Services Discoverability”
25	2015	Palma, et al. [53]	“Are Restful APIs Well-Designed? Detection of Their Linguistic (Anti)Patterns”
26	2015	Rodriguez, et al. [54]	“Assisting Developers to Build High-Quality Code-First Web Service APIs”
27	2015	Sales and Guizzardi [55]	“Ontological Anti-Patterns”
28	2016	Brabra, et al. [56]	“Detecting Cloud (Anti) Patterns: OCCI Perspective”
29	2016	Mateos, et al. [22]	“Keeping Web Service Interface Complexity Low Using an OO Metric-Based Early Approach”
30	2016	Palma, et al. [57]	“Specification and Detection of SOA Antipatterns in Web Services”
31	2016	Wang, et al. [58]	“Bi-Level Identification of Web Service Defects”
32	2017	Chen and Jiang [59]	“Characterizing and Detecting Anti-Patterns in The Logging Code”
33	2017	Ouni, et al. [10]	“Search-Based Web Service Anti Patterns Detection”
34	2017	Pismag [60]	“Prediction of Web Service Anti Patterns Using Machine Learning”
35	2017	Sabir, et al. [61]	“A Lightweight Approach for Specification and Detection of SOAP Anti-Patterns”
36	2017	Velioğlu and Selçuk [62]	“An Automated Code Smell and Anti-Pattern Detection Approach”
37	2017	Wang, et al. [20]	“Interactive Refactoring of Web Service Interfaces Using Computational Search”
38	2018	Arunachalam, et al. [63]	“A Semi Markov Process-Based Web Service Recommendation for Anti-Patterns Detection Using P-EA Algorithm”
39	2018	Blouin, et al. [64]	“User Interface Design Smell: Automatic Detection and Refactoring of Blob Listeners”
40	2018	Fakhoury, et al. [65]	“Keep It Simple: Is Deep Learning Good for Linguistic Smell Detection?”
41	2018	Hirsch, et al. [6]	“Spotting and Removing WSDL Anti-Pattern Root Causes in Code-First Web Services Using NLP Techniques: A Thorough Validation of Impact on Service Discoverability”
42	2018	Kalyani and Vasundra [66]	“Search-Based Web Service and Business Process Anti Pattern Detection”
43	2018	Kumar and Sureka [19]	“An Empirical Analysis on Web Service Anti-Pattern Detection Using a Machine Learning Framework”
44	2018	Ouni, et al. [67]	“A Hybrid Approach for Improving the Design Quality of Web Service Interfaces”
45	2018	Palma, et al. [68]	“UNIDOSA: The Unified Specification and Detection of Service Antipatterns”
46	2019	Brabra, et al. [69]	“On Semantic Detection of Cloud API (Anti)Patterns”
47	2019	Sabir, et al. [70]	“A Systematic Literature Review on The Detection of Smells and their Evolution in Object-Oriented and Service-Oriented Systems”
48	2019	Saluja and Batra [71]	“Assessing Quality by Anti-Pattern Detection in Web Services”
49	2019	Saluja and Batra [72]	“Optimized approach for antipattern detection in service computing architecture”
50	2019	Bogner, et al. [73]	“Towards a Collaborative Repository for the Documentation of Service-Based Antipatterns and Bad Smells”
51	2019	Mateos, et al. [74]	“COBOL Systems Migration to SOA: Assessing Antipatterns and Complexity”
52	2019	Belkhir, et al. [75]	“An observational study on the state of REST API uses in Android mobile applications”

### 3. RESULTS

This section explains the findings of the scoping review. The results are discussed based on the sequent of the RQs.

### 3.1 What are the types of anti-pattern found in SOAP web service?

Web Service Description Language (WSDL) documents are one of the main components of SOAP web service. Usually, the developers designed web service interfaces that can minimize programming efforts, which caused lack in description of the WSDL documents [36],[76]. Past studies reported that improper description in WSDL documents caused issues in the web service discovery [36]. Therefore, web service developers should improve the way WSDL documents are designed, in order to help the web service requesters to discover and understand the web service functionality with minimal efforts [43],[77]. Developers follow one of the approaches in developing web service; (1) contract-first, and (2) code-first [50]. In the contract-first, development of a web service is started from a WSDL definition, which developers specify what the documents would represent. In this approach, WSDL definition is defined using web service standards; the web service implementation is done after that according to this contract [77]. Meanwhile, in the code-first mechanism, development of web service is started from code; hence, the developers can write a web service without the need of knowing how the WSDL document is generated.

The contract-first mechanism is based on designing web service when developers consider a catalog to avoid anti-patterns occurrences which cause complexity and ambiguity in WSDL document [21]. The results if the scoping review suggested twenty-three SOAP anti-patterns as in the subsequence paragraphs.

**1. Ambiguous names:** This anti-pattern occurs when the web service developers use vague, meaningless or unclear words for identifying the main components of WSDL document (i.e., port-type, operations, messages, and part elements) [30]. It is considered as one the most common anti-pattern occurrences in SOAP web services [78]. From a service requesters' perspective, a representative name should describe and confirm the semantics of an element; then meaningless names should be avoided [38]. This major consequences of this type of anti-pattern is that it decreases the opportunity of web service being discovered and understood which in turn prevents usage [51].

**2. Empty messages:** This anti-pattern occurs when web service developers declare a method that does not contain any outputs and does not receive any inputs [52]. Simply said, empty message anti-pattern is public methods that refer to a class that implements a service which does not receive any input parameter [42].

**3. Enclosed data model:** This anti-pattern occurs when the input and output data types of eXtensible Markup Language Schema Definition (XSD) are specified in the WSDL document rather than being written in a separate file. It consequently disallows reuse of data type by other web services very difficult or impossible [50].

**4. Low cohesive operations in the same port-type:** This anti-pattern occurs when web service developers defined two or more operations in a same port-type [57]. For example, checking the availability of the service the function of the service a single port-type. A web service becomes less cohesive, when operations belong to one port-type; however, they do not represent a set of semantically relevant operations [56]. On the other hand, WSDL document with high cohesive operations holds contain a representative service. WSDL documents which have higher number of low cohesive operations might lead to more occurrences of ambiguous name anti-patterns [54].

**5. Redundant data models:** This anti-pattern occurs when the developers use defect WSDL generation tools [6]. It may also happen when the developers used different data types for representing duplicated objects or defining the same data type two or more times (i.e., two data type definitions stand for the same exchangeable information) in a WSDL document [40],[47]. The redundant data models cause confusing in understanding WSDL documents [36].

**6. Whatever types:** This anti-pattern may occur when the developers define uncommon data types or unsupported data types (i.e., the data types that do not belong to primitive and standard data types) [77],[79]. When the developers define data with any type (i.e., Whatever data type), then the standard data type will be implemented inside WSDL document as `xsd:any` [21],[80]. `xsd:any` may appear in WSDL document because of inefficient or defective WSDL generation tools [50],[52]. Whatever data types negatively affect the web service usability as it hinders understandability [81].

**7. Inappropriate or lacking comments:** This anti-pattern occurs when the developers do not write appropriate comments in WSDL documents [50]. Many WSDL documents suffered from inappropriate or lacking comments [36]. A well-documented WSDL document has concise explanatory comment for each operation that describes the semantic of the offered functions [36]. Consequently, inappropriate or lacking comments does not help in revealing the real purpose of the service, which leads to reducing the understandability and the usability of the concerned web service [81].

**8. Redundant port-types:** This anti-pattern occurs when multiple port types present duplicated set of operations in a web service [63]. The developers defined two or more port types that offer a same operation with a same message; however, each port type is restricted to a different transport protocol [36],[54]. Consequently, the redundant port-types causes unnecessarily confusion in understanding WSDL documents [36].

**9. Undercover fault information within standard messages:** This anti-pattern occurs when output messages notify service errors [51],[54]. A service error message should use output messages rather than local SOAP failure

messages [29]. Consequently, the web services return error message within output messages that may negatively influence syntactic registries that exploit either the names or the XML structure of message parts [23].

**10. God object web service (GOWS):** This anti-pattern occurs when web services hold too many operations that belong to variant business abstractions. In this case, the web service is complicated to use due to low cohesion of its methods [10].

**11. Fine-grained web service (FGWS):** This anti-pattern occurs when web services contain few numbers of operations implementing only a part of abstraction [10],[82]. It frequently demands for other web services to complete abstraction, and often overhead (e.g., maintenance, communication) outweigh its utility [67]. FGWS anti-pattern results in higher architectural complexity and reduced web service usability [10],[57].

**12. Chatty web service (CWS):** This anti-pattern occurs when a set of operations are required to execute single abstraction [20],[58]. Consequently, web services often have many compact operations that increase response time; therefore, limiting the web services performance [83].

**13. Data web service (DWS):** This anti-pattern occurs when web services consist of getters and setters' operations (i.e., accessor operation) [83]. DWS anti-pattern in distributed environment refers to web services that only execute data access operations or simple information retrieval [20],[58]. DWS may negatively impact web service usability because it commonly handles small messages of primitive data types with strong data cohesion [10].

**14. CRUD interface (CI):** This anti-pattern occurs when a web service supports remote procedure call [20] that allows Create, Read, Update and Delete (CRUD) operations from databases [10],[58]. The CRUD operations require several methods to be accomplished which increases the complexity of web services [10].

**15. Maybe it is not RPC (MNR):** This anti-pattern occurs when a web service mainly presents CRUD operation for large business entities [82] which consequently requires large number of parameters [10]. MNR anti-pattern causes inefficiency because the users will wait for the synchronous responses from many parameters.

**16. Bloated service:** This anti-pattern occurs when web services hold too many parameters [45]; hence, leads to a complex operations execution with low cohesion between such operations [68]. Consequently, it leads to low web service accessibility, maintainability, testability, and reusability [45].

**17. Duplicated web service:** This anti-pattern occurs when two or more web services contain identical operations with identical names and message operations [61]. It also might

exist when common operations have an identical name or parameters [57].

**18. Nobody home:** This anti-pattern occurs when a web service (or resource) is defined, but the methods of this web service (or resource) are never invoked by service requesters [68]. These web services lead to non-usage [78].

**19. Bottleneck service:** This anti-pattern occurs when a web service is frequently called by service requesters that causes bottleneck of incoming and outgoing requests. It may be used by many service requesters; hence, it increases response time and caused low availability due to heavy traffic [45].

**20. Stovepipe service:** This anti-pattern occurs when many private or protected methods for executing infrastructure and utility operations (i.e., logging, data validation, notifications, etc.) exist in few business processes [45]. It may cause duplicated code in web service, increase the development time, inconsistent operations, and inextensible services.

**21. Service chain:** This anti-pattern occurs when web services requesters requested for consecutive web service invocations to achieve their goals [78]. The consecutive requests affect the subsequent call for the web services [45].

**22. Tiny service/ nano service:** This anti-pattern occurs when small web services with few numbers of methods and operations [78] implements a part of abstraction [68]. Tiny web service often needs many web services to be used together to perform operations [59]. This anti-pattern leads to a more complex process, reduces flexibility, and causes many service-oriented architectures failures [45],[68].

**23. Multi-service:** This anti-pattern is the opposite of tiny service anti-pattern. It is also considered as God Object web service [68]. Multi-service anti-pattern is found in web services that run more than one method for distinct business and technical abstraction [45],[68]. It is described by many low cohesion operations, very high response time, and low availability to service requesters because it is overloaded [68],[78].

In summary, the scoping review suggested twenty-three anti-patterns that commonly found in SOAP web services. These anti-patterns have impact on various aspects of web services including maintainability, efficiency, and discovery.

### 3.2 RQ2: What are the types of anti-pattern found in RESTful web service?

A RESTful provides data by an application programming interface (API) over the Internet through HTTP [84]. Therefore, a good design with proper API name will certainly attract service requesters [53]. Hence, understandability and reusability are the essential aspects in designing and developing RESTful web services. Analysis of

the documents included in the scoping reviews suggested the following anti-patterns for RESTful web service.

**1. Uniform Resource Identifier (URI) design:** This anti-pattern concerned with the practices in designing URI in the web services. URIs should be easy to read and tidy [53],[69]. There are three types of URI anti-patterns:

- i. Amorphous URIs anti-pattern: It occurs if URI consists of capital letters, symbols, and underscore; to name a few. It causes lower readability and understandability of the intended URI [53],[69].
- ii. CRUD URI anti-pattern: It occurs when CRUD is used instead of standard HTTP methods [53],[69]. It overloads the HTTP methods and prevents service requesters to use the appropriate and standard HTTP methods [53],[69].
- iii. Pluralized Nodes anti-pattern: It occurs when plural words are used in PUT/DELETE requests, or singular nouns used in POST request [53],[69]. It may affect the server response to the web service requests.

**2. HTTP methods:** This anti-pattern concerned with using HTTP methods in the right context [69]. There are two scenarios of HTTP methods anti-pattern;

- i. Tunneling through GET: It occurs when the developers depend on GET method to execute operations for creating, deleting or updating resources [48],[69],[78]. GET can be used for other actions except for accessing resources [48].
- ii. Tunneling through POST: It occurs when the developers depend on POST method to execute operations on creating, deleting or updating resources [48],[69],[78]. It is very similar to tunneling through GET; however, in addition to the URI, the POST requests can be used for operations and parameters to access resources [48]. These anti-patterns may cause breaks in the semantic purpose of each HTTP methods [69].

**3. Error handling:** This anti-pattern concerned with HTTP request methods for error handling and how they must be used as a response of HTTP messages [69]. The developers tend to avoid the application-level status code (e.g., 200, 404, and 500, use the wrong status code, or may not use any status code) despite a standard error message code [68],[78].

**4. HTTP header:** This anti-pattern concerned with the practices in designing HTTP headers. There are two common RESTful anti-patterns in the design of HTTP headers:

- i. Ignoring caching: It occurs when the developers and service requesters avoid the use of the caching capability due to the complexity of its implementation [48]. Consequently, it causes a decreasing of scalability in request per second, which degrades the overall performance [69].
- ii. Ignoring Multipurpose Internet Mail Extensions (MIME): It occurs when a server uses personalized formats or depends on a unique representation [48],[68]. Consequently, it limits usability (accessibility), reusability, and limits the resources readability [69],[78].

**5. Hypermedia:** This anti-pattern occurs when the developers forgot to link the resources together [69]. The absence of Uniform Resource Locator (URL) links between resources disallows the requesters to follow the links because the server does not provide it [78]. Consequently, this anti-pattern decreases the dynamic interaction between the service requesters and servers that leads to low usability and accessibility [48],[68].

**6. Linguistic:** This anti-pattern occurs when the developers have poor linguistic practices in designing web services. It leads to ambiguous detailed description [53]. There are four scenarios of linguistic anti-pattern:

- i. Contextless recourse name: It occurs when URI consists many nodes referring to different context [78]. Consequently, it causes a decrease in the understandability and usability of the web service [53].
- ii. Non-hierarchical nodes: It occurs when the nodes in a URI are not hierarchically designed [78]. It appears when at least one node in URI is not linked to its neighbor node [53]. Consequently, it may confuse the service requesters and hinder the real purpose of the web services, and reduce their understandability and usability [53],[78].

**7. Breaking self-descriptiveness:** This anti-pattern occurs when the developers overlook the standardized header, protocol, or format, and use customized ones [48],[70]. This anti-pattern wipes out the role of a message header and also limits the adaptability and reusability of web service resources [78].

**8. Misusing cookies:** This anti-pattern concerned with the use of disallowed session states at the server [70]. An example of misusing cookies is the use of keys or tokens in the Set Cookie or Cookie header field of server-side session [78].

**9. Missing query interface:** This anti-pattern occurs when the developers ignore to provide support to query interface on all requests [69],[78] which disables the requesters to discover all capabilities of a web service [56],[69].

**10. Ambiguous name:** This anti-pattern occurs when the developers used weak naming for interface elements such as using the long or short identifiers, use of operations that are not published syntactically and semantically, and use general terms as identifiers [48],[78]. Consequently, ambiguous name negatively affects the discoverability and usability of the web services [48].

**11. Bloated service:** This anti-pattern occurs when a web service contains too many parameters [45]; hence, leads to a complex operations execution with low cohesion between such operations [68]. Consequently, it leads to low web service accessibility, maintainability, testability, and reusability [45].

**12. Deprecated resource:** This anti-pattern occurs when a service is created but it is not being used by the clients [78]. Consequently, the web service will not be invoked, even though it may be combined with other services.


In summary, the scoping review suggested twelve anti-patterns that commonly found in RESTful web services. These anti-patterns have impact on various aspects of web services including maintainability, efficiency, and discovery.

### 3.3 What are the common anti-patterns of SOAP and RESTful?

The results of the scoping review for RQ1 and RQ2 suggested that SOAP and RESTful web services suffered from multiple anti-pattern problems. Based on the analysis, there are a few common anti-patterns found in both protocols, while many of them found in RESTful or SOAP separately. Figure 2 listed the web services anti-patterns for SOAP and REST. Only two types of anti-patterns were common to the two web service protocols namely ambiguous names and bloated services.

<u>SOAP</u>	<u>RESTful</u>
▪ Ambiguous names	▪ Uniform Resource Identifier (URI) design
▪ Empty messages	▪ HTTP methods
▪ Enclosed data model	▪ Error handling
▪ Low cohesive operations in the same port-type	▪ HTTP header
▪ Redundant data models	▪ Hypermedia
▪ Whatever types	▪ Linguistic
▪ Inappropriate or lacking comments	▪ Breaking self-descriptiveness
▪ Redundant port-types	▪ Misusing cookies
▪ Undercover fault information within standard messages	▪ Missing query interface
▪ God object web service	▪ Ambiguous name
▪ Fine-grained web service	▪ Bloated service
▪ Chatty web service	▪ Deprecated resource
▪ Data web service	
▪ CRUD interface	
▪ Maybe it is not RPC	
▪ Bloated service	
▪ Duplicated web service	
▪ Nobody home	
▪ Bottleneck service	
▪ Stovepipe service	
▪ Service chain	
▪ Tiny service/ nano service	
▪ Multi-service	



- Ambiguous name
- Bloated service

**Figure 2:** The types of anti-patterns found in SOAP and RESTful web services

## 4. CONCLUSION

This paper reported a scoping review of web service anti-patterns found in two prominent protocols of SOAP and RESTful. The results of the study suggested twenty-three anti-patterns in SOAP and twelve anti-patterns in RESTful. Only a few anti-patterns were found common in both protocols. The results of the study may be beneficial for web service developers in understanding the possible anti-patterns that may occurs in designing web services. It can assist them by avoiding such anti-patterns; hence, produce a good web

service that is easy to maintained and possible reuse by other applications. In future, we aim at studying the techniques to mitigate such anti-patterns.

## ACKNOWLEDGMENT

The authors thank the Ministry of Higher Education Malaysia in funding this study under the Trans-Disciplinary Research Grant Scheme (Ref: TRGS/2/2014/ UUM/01/3/4, UUM S/O Code:13170), and Research and Innovation Management Centre, Universiti Utara Malaysia for the administration of this study.

## REFERENCES

1. P. El-Kafrawy, E. Elabd, and H. Fathi. **A trustworthy reputation approach for web service discovery**, *Procedia Computer Science*, vol. 65, pp. 572-581, 2015. <https://doi.org/10.1016/j.procs.2015.09.001>
2. W. J. Obidallah and B. Raahemi. **A Taxonomy to Characterize Web Service Discovery Approaches, Looking at Five Perspectives**, in *2016 IEEE Symposium on Service-Oriented System Engineering (SOSE)*, 2016, pp. 458-459. <https://doi.org/10.1109/SOSE.2016.13>
3. I. Lizarralde, J. M. Rodriguez, C. Mateos, and A. Zunino. **Word embeddings for improving REST services discoverability**, in *2017 XLIII Latin American Computer Conference (CLEI)*, 2017, pp. 1-8. <https://doi.org/10.1109/CLEI.2017.8226444>
4. J. Wang, P. Gao, Y. Ma, K. He, and P. C. Hung. **A web service discovery approach based on common topic groups extraction**, *IEEE Access*, vol. 5, pp. 10193-10208, 2017. <https://doi.org/10.1109/ACCESS.2017.2712744>
5. M. Curiel and A. Pont. **Workload generators for web-based systems: Characteristics, current status, and challenges**, *IEEE Communications Surveys & Tutorials*, vol. 20, pp. 1526-1546, 2018. <https://doi.org/10.1109/COMST.2018.2798641>
6. M. Hirsch, A. Rodriguez, J. M. Rodriguez, C. Mateos, and A. Zunino. **Spotting and Removing WSDL Anti-pattern Root Causes in Code-first Web Services Using NLP Techniques: A Thorough Validation of Impact on Service Discoverability**, *Computer Standards & Interfaces*, vol. 56, pp. 116-133, 2018.
7. M. H. I. Hamzah, F. Baharom, and H. Mohd. **An exploratory study for investigating the issues and current practices of Service-Oriented Architecture adoption**, *Journal of Information and Communication Technology*, vol. 18, pp. 273-304, 2019.
8. M. H. I. Hamzah, F. Baharom, and H. Mohd. **A Service-Oriented Architecture Adoption Maturity Matrix using Kano Model: Cross Evaluation between IT and Business Benefits**, *Journal of Telecommunication, Electronic and Computer Engineering (JTEC)*, vol. 9, pp. 105-112, 2017.
9. W. Dai, V. Vyatkin, J. H. Christensen, and V. N. Dubinin. **Bridging service-oriented architecture and**

- IEC 61499 for flexibility and interoperability**, *IEEE Transactions on Industrial Informatics*, vol. 11, pp. 771-781, 2015.
10. A. Ouni, M. Kessentini, K. Inoue, and M. Ó. Cinnéide. **Search-Based Web Service Antipatterns Detection**, *IEEE Transactions on Services Computing*, vol. 10, pp. 603-617, 2017.
  11. B. Saravana Balaji, R. S. Rajkumar, and B. F. Ibrahim. **Service profile based ontological system for selection and ranking of business process web services**, *International Journal of Advanced Trends in Computer Science and Engineering*, vol. 8, pp. 18-22, 2019. <https://doi.org/10.30534/ijatcse/2019/04812019>
  12. M. G. Galety, B. Saravana Balaji, and M. S. Saleem Basha. **OSSR-P: Ontological service searching and ranking system for PaaS services**, *International Journal of Advanced Trends in Computer Science and Engineering*, vol. 8, pp. 271-276, 2019.
  13. R. B. Brinhosa, C. M. Westphall, C. B. Westphall, D. R. Dos Santos, F. Grezele, and D. R. Westphall. **A validation model of data input for web services**, in *Twelfth International Conference on Networks*, 2013, pp. 87-94.
  14. J. Yu, Q. Z. Sheng, J. K. Swee, J. Han, C. Liu, and T. H. Noor. **Model-driven development of adaptive web service processes with aspects and rules**, *Journal of Computer and System Sciences*, vol. 81, pp. 533-552, 2015. <https://doi.org/10.1016/j.jcss.2014.11.008>
  15. A. L. Lemos, F. Daniel, and B. Benatallah. **Web service composition: a survey of techniques and tools**, *ACM Computing Surveys (CSUR)*, vol. 48, p. 33, 2016. <https://doi.org/10.1145/2831270>
  16. R. Gunasri and R. Kanagaraj. **Natural Language Processing and Clustering based service discovery**, *International Journal of Scientific & Technology Research*, vol. 3, pp. 28-31, 2014.
  17. A. De Renzis, M. Garriga, A. Flores, A. Cechich, C. Mateos, and A. Zunino. **A domain independent readability metric for web service descriptions**, *Computer Standards & Interfaces*, vol. 50, pp. 124-141, 2017. <https://doi.org/10.1016/j.csi.2016.09.005>
  18. T. Masood, A. Nadeem, and S. Ali. **An automated approach to regression testing of web services based on WSDL operation changes**, in *2013 IEEE 9th International Conference on Emerging Technologies (ICET)*, 2013, pp. 1-5.
  19. L. Kumar and A. Sureka. **An Empirical Analysis on Web Service Anti-pattern Detection Using a Machine Learning Framework**, in *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, 2018, pp. 2-11. <https://doi.org/10.1109/COMPSAC.2018.00010>
  20. H. Wang, M. Kessentini, and A. Ouni. **Interactive refactoring of web service interfaces using computational search**, *IEEE Transactions on Services Computing*, 2017.
  21. J. L. O. Coscia, M. Crasso, C. Mateos, and A. Zunino. **An approach to improve code-first web services discoverability at development time**, in *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, 2012, pp. 638-643.
  22. C. Mateos, A. Zunino, S. Misra, D. Anabalon, and A. Flores. **Keeping web service interface complexity low using an oo metric-based early approach**, in *2016 XLII Latin American Computing Conference (CLEI)*, 2016, pp. 1-12. <https://doi.org/10.1109/CLEI.2016.7833366>
  23. A. Siswanto, N. Katuk, and K. R. Ku-Mahamud. **Fingerprint Template Protection Schemes: A Literature Review**, *Journal of Theoretical & Applied Information Technology*, vol. 96, 2018.
  24. N. Katuk, K. R. Ku-Mahamud, and N. H. Zakaria. **A Review of The Current Trends and Future Directions of Camera Barcode Reading**, *Journal of Theoretical and Applied Information Technology*, vol. 97, pp. 2268-2288, 2019.
  25. M. T. Pham, A. Rajić, J. D. Greig, J. M. Sargeant, A. Papadopoulos, and S. A. McEwen. **A scoping review of scoping reviews: advancing the approach and enhancing the consistency**, *Research synthesis methods*, vol. 5, pp. 371-385, 2014. <https://doi.org/10.1002/jrsm.1123>
  26. H. Arksey and L. O'Malley. **Scoping studies: towards a methodological framework**, *International journal of social research methodology*, vol. 8, pp. 19-32, 2005.
  27. R. Armstrong, B. J. Hall, J. Doyle, and E. Waters. **'Scoping the scope' of a cochrane review**, *Journal of Public Health*, vol. 33, pp. 147-150, 2011. <https://doi.org/10.1093/pubmed/fdr015>
  28. J. O'Flaherty and C. Phillips. **The use of flipped classrooms in higher education: A scoping review**, *The internet and higher education*, vol. 25, pp. 85-95, 2015.
  29. N. Kokash. **A comparison of web service interface similarity measures**, in *STAIRS*, 2006, pp. 220-231.
  30. Y. Zheng and P. Krause. **Asynchronous semantics and anti-patterns for interacting web services**, in *2006 Sixth International Conference on Quality Software (QSIC'06)*, 2006, pp. 74-84.
  31. J. He and I. Yen. **Adaptive User Interface Generation for Web Services**, in *IEEE International Conference on e-Business Engineering (ICEBE'07)*, 2007, pp. 536-539.
  32. J. Beaton, S. Y. Jeong, Y. Xie, J. Stylos, and B. A. Myers. **Usability challenges for enterprise service-oriented architecture APIs**, in *2008 IEEE Symposium on Visual Languages and Human-Centric Computing*, 2008, pp. 193-196. <https://doi.org/10.1109/VLHCC.2008.4639084>
  33. H. Hacıgümüş. **Anti-Patterns: Integrating Distributed and Heterogeneous Data Sources in SOAs**, in *2008 IEEE Congress on Services-Part I*, 2008, pp. 95-96.
  34. J. Král and M. Žemlicka. **Popular SOA antipatterns**, in *2009 Computation World: Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns*, 2009, pp. 271-276.
  35. M. Crasso, J. M. Rodriguez, A. Zunino, and M. Campo. **Revising WSDL documents: Why and how**, *IEEE Internet Computing*, vol. 14, pp. 48-56, 2010. <https://doi.org/10.1109/MIC.2010.81>



36. J. M. Rodriguez, M. Crasso, A. Zunino, and M. Campo. **Improving Web Service descriptions for effective service discovery**, *Science of Computer Programming*, vol. 75, pp. 1001-1021, 2010.
37. J. M. Rodriguez, M. Crasso, C. Mateos, A. Zunino, and M. Campo. **The EasySOC project: A rich catalog of best practices for developing Web Service applications**, in *2010 XXIX International Conference of the Chilean Computer Science Society*, 2010, pp. 33-42.
38. J. M. Rodriguez, M. Crasso, A. Zunino, and M. Campo. **Automatically detecting opportunities for web service descriptions improvement**, in *Conference on e-Business, e-Services and e-Society*, 2010, pp. 139-150. [https://doi.org/10.1007/978-3-642-16283-1\\_18](https://doi.org/10.1007/978-3-642-16283-1_18)
39. S. Batra and S. Bawa. **Semantic discovery of web services using principal component analysis**, *International Journal of Physical Sciences*, vol. 6, pp. 4466-4472, 2011.
40. C. Mateos, M. Crasso, A. Zunino, and J. L. O. Coscia. **Detecting WSDL bad practices in code-first Web Services**, *International Journal of Web and Grid Services*, vol. 7, p. 357, 2011. <https://doi.org/10.1504/IJWGS.2011.044710>
41. J. M. Rodriguez, M. Crasso, C. Mateos, A. Zunino, and M. Campo. **Bottom-up and top-down cobol system migration to web services**, *IEEE Internet Computing*, vol. 17, pp. 44-51, 2011.
42. C. Mateos, M. Crasso, A. Zunino, and J. L. O. Coscia. **Avoiding WSDL bad practices in code-first web services**, *Electronic Journal of SADIO (EJS)*, vol. 11, pp. 31-48, 2012.
43. J. L. O. Coscia, C. Mateos, M. Crasso, and A. Zunino. **Anti-pattern free code-first web services for state-of-the-art Java WSDL generation tools**, 2013.
44. Z. Han, P. Gong, L. Zhang, J. Ling, and W. Huang. **Definition and detection of control-flow anti-patterns in process models**, in *2013 IEEE 37th Annual Computer Software and Applications Conference Workshops*, 2013, pp. 433-438. <https://doi.org/10.1109/COMPSACW.2013.111>
45. F. Palma, M. Nayrolles, N. Moha, Y.-G. Guéhéneuc, B. Baudry, and J.-M. Jézéquel. **SOA Antipatterns: An Approach for Their Specification and Detection**, *International Journal of Cooperative Information Systems*, vol. 22, p. 1341004, 2013.
46. J. M. Rodriguez, M. Crasso, and A. Zunino. **An approach for web service discoverability anti-pattern detection for journal of web engineering**, *Journal of Web Engineering*, vol. 12, pp. 131-158, 2013.
47. J. L. O. Coscia, C. Mateos, M. Crasso, and A. Zunino. **Refactoring code-first Web Services for early avoiding WSDL anti-patterns: Approach and comprehensive assessment**, *Science of Computer Programming*, vol. 89, pp. 374-407, 2014/09/01/ 2014. <https://doi.org/10.1016/j.scico.2014.03.015>
48. F. Palma, J. Dubois, N. Moha, and Y.-G. Guéhéneuc. **Detection of REST patterns and antipatterns: a heuristics-based approach**, in *International Conference on Service-Oriented Computing*, 2014, pp. 230-244. [https://doi.org/10.1007/978-3-662-45391-9\\_16](https://doi.org/10.1007/978-3-662-45391-9_16)
49. M. Torkamani and H. Bagheri. **A Systematic Method for Identification of Anti-patterns in Service Oriented System Development**, *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 4, pp. 2088-8708, 02/01 2014.
50. C. Mateos, M. Crasso, J. M. Rodriguez, A. Zunino, and M. Campo. **Measuring the impact of the approach to migration in the quality of web service interfaces**, *Enterprise Information Systems*, vol. 9, pp. 58-85, 2015.
51. C. Mateos, J. M. Rodriguez, and A. Zunino. **A tool to improve code-first Web services discoverability through text mining techniques**, *Software: Practice and Experience*, vol. 45, pp. 925-948, 2015. <https://doi.org/10.1002/spe.2268>
52. C. Mateos, M. Crasso, A. Zunino, and J. L. O. Coscia. **A Stitch in Time Saves Nine: Early Improving Code-First Web Services Discoverability**, *International Journal of Cooperative Information Systems*, vol. 24, p. 1550004, 2015. <https://doi.org/10.1142/S0218843015500045>
53. F. Palma, J. Gonzalez-huerta, and N. Moha. **Are RESTful APIs Well-designed? Detection of their Linguistic ( Anti ) Patterns**, in *Int. Conf. Serv. Comput*, 2015, pp. 171-187. [https://doi.org/10.1007/978-3-662-48616-0\\_11](https://doi.org/10.1007/978-3-662-48616-0_11)
54. J. M. Rodriguez, C. Mateos, and A. Zunino. **Assisting developers to build high-quality code-first Web Service APIs**, *J. Web Eng.*, vol. 14, pp. 251-285, 2015.
55. T. P. Sales and G. Guizzardi. **Ontological anti-patterns**, *Data Knowl. Eng.*, vol. 99, pp. 72-104, 2015.
56. H. Brabra, A. Mtibaa, L. Sliman, W. Gaaloul, F. Gargouri, and B. Benatallah. **Detecting cloud (anti)patterns: OCCI perspective**, in *ICSOC 2016 : 14th International Conference on Service-Oriented Computing*, Banff, Canada, 2016, pp. 202 - 218. [https://doi.org/10.1007/978-3-319-46295-0\\_13](https://doi.org/10.1007/978-3-319-46295-0_13)
57. F. Palma, N. Moha, and G. Tremblay. **Specification and Detection of SOA Antipatterns in Web Services**, in *Eur. Conf. Softw. Archit.*, 2014, pp. 58-73.
58. H. Wang, M. Kessentini, and A. Ouni. **Bi-level identification of web service defects**, in *International Conference on Service-Oriented Computing*, 2016, October, pp. 352-368.
59. B. Chen and Z. M. Jiang. **Characterizing and Detecting Anti-Patterns in the Logging Code**, in *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*, 2017, pp. 71-81.
60. V. K. J. Pismag. **Prediction of Web Service Antipatterns Using Machine Learning**, Master of Science (Software Engineering), University of Michigan - Dearborn, 2017.
61. F. Sabir, G. Rasool, and M. Yousaf. **A Lightweight Approach for Specification and Detection of SOAP Anti-Patterns.**, *International Journal of Advanced Computer Science and Applications*, vol. 8, pp. 455-467, 2017. <https://doi.org/10.14569/IJACSA.2017.080555>
62. S. Velioglu and Y. E. Selçuk. **An automated code smell and anti-pattern detection approach**, in *2017 IEEE 15th International Conference on Software Engineering*

- Research, Management and Applications (SERA)*, 2017, pp. 271-275.  
<https://doi.org/10.1109/SERA.2017.7965737>
63. N. Arunachalam, D. Cousalya, and P. Subathra. **A Semi-Markov process based web service recommendation for anti-pattern detection using P-EA algorithm**, *International Journal of Pure and Applied Mathematics*, vol. 119, pp. 1693-1701, 2018.
64. A. Blouin, V. Lelli, B. Baudry, and F. Coulon. **User interface design smell: Automatic detection and refactoring of Blob listeners**, *Information and Software Technology*, vol. 102, pp. 49-64, 2018.
65. S. Fakhoury, V. Arnaoudova, C. Noiseux, F. Khomh, and G. Antoniol. **Keep it simple: Is deep learning good for linguistic smell detection?**, in *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2018, pp. 602-611.  
<https://doi.org/10.1109/SANER.2018.8330265>
66. G. K. Kalyani and S. Vasundra. **Search Based Web Service and Business Process Anti Pattern Detection**, *International Journal for Research in Engineering Application & Management (IJREAM)*, vol. 4, pp. 558-561, 2018.
67. A. Ouni, H. Wang, M. Kessentini, S. Bouktif, and K. Inoue. **A Hybrid Approach for Improving the Design Quality of Web Service Interfaces**, *ACM Trans. Internet Technol.*, vol. 19, pp. 1-24, 2018.
68. F. Palma, N. Moha, and Y. Gu. **UniDoSA: The Unified Specification and Detection of Service Antipatterns**, *IEEE Transactions on Software Engineering*, pp. 1-1, 2018.
69. H. Brabra, A. Mtibaa, F. Petrillo, P. Merle, L. Sliman, N. Moha, *et al.* **On Semantic Detection of Cloud API (Anti)Patterns**, *Information and Software Technology*, vol. 107, pp. 65-82, 2019.  
<https://doi.org/10.1016/j.infsof.2018.10.012>
70. F. Sabir, F. Palma, G. Rasool, Y.-G. Guéhéneuc, and N. Moha. **A systematic literature review on the detection of smells and their evolution in object-oriented and service-oriented systems**, *Software: Practice and Experience*, vol. 49, pp. 3-39, 2019.  
<https://doi.org/10.1002/spe.2639>
71. S. Saluja and U. Batra. **Assessing Quality by Anti-pattern Detection in Web Services**, *SSRN Electron. J.*, pp. 47-52, 2019.
72. S. Saluja and U. Batra. **Optimized approach for antipattern detection in service computing architecture**, *Journal of Information and Optimization Sciences*, vol. 40, pp. 1069-1080, 2019.  
<https://doi.org/10.1080/02522667.2019.1638000>
73. J. Bogner, T. Bocek, M. Popp, D. Tschelchlov, S. Wagner, and A. Zimmermann. **Towards a Collaborative Repository for the Documentation of Service-Based Antipatterns and Bad Smells**, in *2019 IEEE International Conference on Software Architecture Companion (ICSA-C)*, 2019, pp. 95-101.
74. C. Mateos, A. Zunino, A. Flores, and S. Misra. **COBOL Systems Migration to SOA: Assessing Antipatterns and Complexity**, *Information Technology and Control*, vol. 48, pp. 71-89, 2019.
75. A. Belkhir, M. Abdellatif, R. Tighilt, N. Moha, Y.-G. Guéhéneuc, and É. Beaudry. **An observational study on the state of REST API uses in Android mobile applications**, in *Proceedings of the 6th International Conference on Mobile Software Engineering and Systems*, 2019, pp. 66-75.
76. J. Pasley. **Avoid XML schema wildcards for Web service interfaces**, *IEEE Internet Computing*, vol. 10, pp. 72-79, 2006.
77. F. Alshraideh, S. Hanna, and R. Alazaidah. **An approach to extend WSDL-based data types specification to enhance web services understandability**, *International Journal of Advanced Computer Science and Applications*, vol. 6, pp. 88-98, 2015.  
<https://doi.org/10.14569/IJACSA.2015.060314>
78. F. Palma. **Unifying Service Oriented Technologies for the Specification and Detection of their Antipatterns**, Doctoral dissertation, École Polytechnique de Montréal, 2015.
79. F. Alshraideh and N. Katuk. **Enrichment of Data Type Specification for Web Service Compatibility**, in *The 3rd Innovation and Analytics Conference & Exhibition(IACE) 2016*, 2016.
80. C. Mateos, M. Crasso, A. Zunino, and J. L. O. Coscia. **Revising WSDL documents: Why and how, Part 2**, *IEEE Internet Computing*, vol. 17, pp. 46-53, 2013.
81. Y.-H. Wang and I.-C. Wu. **Achieving high and consistent rendering performance of Java AWT/Swing on multiple platforms**, *Software: Practice and Experience*, vol. 39, pp. 701-736, 2009.
82. H. Wang. **Intelligent Web Services Architecture Evolution Via An Automated Learning-Based Refactoring Framework**, Doctor of Philosophy, University of Michigan Dearborn, 2018.
83. A. Ouni, R. G. Kula, M. Kessentini, and K. Inoue. **Web Service Antipatterns Detection Using Genetic Programming**, presented at the Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation, Madrid, Spain, 2015.
84. A. Arcuri. **RESTful API Automated Test Case Generation with EvoMaster**, *ACM Trans. Softw. Eng. Methodol.*, vol. 28, pp. 1-37, 2019.  
<https://doi.org/10.1145/3293455>