A Lightweight Approach for Specification and Detection of SOAP Anti-Patterns

Fatima Sabir, Ghulam Rasool, Maria Yousaf
Department of Computer Science
COMSATS Institute of Information Technology
Defence Road, off Raiwind Road, Lahore
Pakistan

***Abstract*—Web-services have become a governing technology for Service Oriented Architectures due to reusability of services and their dependence on other services. The evolution in service based systems demands frequent changes to provide quality of service to customers. It is realised by different researchers that evolution in service based systems may degrade design and quality of service and may generate poor solutions known as anti-patterns. The detection of anti-patterns from web services is an important research realm and it is continuously getting the attention of researchers. There are a number of techniques and tools presented for detection of anti-patterns from object oriented software applications but only few approaches are presented for detection of anti-patterns from SOA. The state of the art anti-pattern detection approaches presented for detection of anti-patterns from SOA are not flexible enough and they are limited to detection of only a few anti-patterns. We present a flexible approach supplemented with a tool support to detect 10 anti-patterns from different SOA-based applications. We compare results of our approach with two representative state of the art approaches.**

Keywords—SOAP web services; Anti-patterns; Bad smells; SQL

I. Introduction

Design patterns suggest viable solutions to the problems that occur again and again in the design of the software [1]. Design patterns follow the fundamental design principles for the development of software applications. Anti-patterns violate fundamental design principles and they are poor solutions adopted by developers due to deadline pressure, lack of awareness and time to market constraints. Anti-patterns may have a negative impact on the quality and performance of software applications and their presence may result in degrading the structure of the services [2]. The identification of anti-patterns from the web services is a primary step for the removal of anti-patterns from service based systems. It is important to have knowledge about the presence of anti-patterns in the software systems because it helps to improve the software at its abstraction level. It is reported through different studies that timely detection and correction of anti-patterns from software systems improve system performance and quality [4, 7]. This edge motivates researchers to offer assistance for unskilled designers through the detection of anti-patterns.

Service Oriented Architecture (SOA) is an arising architecture paradigm that is widely adopted by software industry for the development of distributed and heterogeneous applications. SOA allows the growth of timely, cost effective, flexible, adaptable, reusable, scalable and extendable distributed software applications with enhanced security by composing services through independent, reusable, and platform independent software modules that are easy to get via a network [8]. The application of SOA for emerging technologies such as cloud computing, big data and mobile applications is continuously escalating. Web-services have become an important technology for Service Oriented Architectures for the development of Service Based Systems (SBS) such as Amazon, Google, eBay, PayPal, Facebook, Dropbox, etc. Service based systems need to evolve with time to fulfill requirements of users. These systems also evolve to accommodate new execution contexts such as addition of new technologies, devices and products [8]. The evolution of service based systems may degrade design and quality of services and it may also cause the appearance of common poor solutions called anti-patterns. These anti-patterns affect the quality of service and can hinder maintenance and evolution. It reflects from state of the art anti-pattern detection techniques that mostly the concentration was on the static analysis of Web-services or on anti-patterns in other Service Oriented Architecture technologies (e.g., Service Component Architecture) [9].

It has been reported that anti-patterns have impact on the progress and maintenance of software systems [10]. The motivation for automatic identification of SOA related anti-patterns is to improve the quality of service and to make maintenance and evolution easy. Maintaining changes in web services is a common practice to provide quality of services to the users. A study has shown that the software maintenance requires eighty to ninety per cent of the total budget in its whole life cycle [11]. Most state of the art techniques focused on detection of anti-patterns from object oriented software applications but these techniques are not capable of detecting anti-patterns from SOA. We identified only a few representative approaches on specification and detection of SOA anti-patterns from Web-services [9, 12, 13, 14, 15, 16, 18]. To the best of our knowledge, most authors used different metrics and static/ dynamic analysis methods for the detection of anti-patterns from web-services. The state of the art anti-pattern detection approaches has some limitations: SODA-W [13] approach detects anti-patterns by just considering interface level metrics and it ignores implementation details. PA-E [14] approach detects anti-patterns from web services by considering their classes as well as implementation details but it is not capable to identify classes that create a problem at the interface level. Moreover, low cohesion operation and duplicated web service anti-patterns are not detected by this approach. The both approaches are also not able to identify the location of defected code segments that play a major role for interface level implementation. There are also no standard definitions for web-service anti-patterns that are important for their accurate detection. Moreover, there are still no standard benchmark systems for comparing and evaluating results of anti-pattern detection techniques for SOA.

The proposed approach is flexible and extendable to implement code first concept. The presented approach is free from the implementation restrictions of WSDL interface. SOAP services might be implemented by using multiple languages such as C#, Java, Perl etc. The approach may detect anti-patterns from WSDL interface of web services as well as source code due to support of multiple languages. The proposed approach is supplemented with a tool support that is used to detect 10 SOA anti-patterns from different Web-services. The objective of presented approach is to analyse the structure and quality of Web-services and automatically identify anti-patterns that may help the progression and growth of Web-services. The proposed approach is implemented by using C# dot.net Framework. We also focus on improving the accuracy in comparison to existing methods available for the detection of Web-service specific anti-patterns.

Following are the major contributions of our work:

- Standardised definitions of 10 Web-services related anti-patterns.
- A flexible and scalable approach supplemented with a tool support for the detection of 10 anti-patterns from different Web-services.
- Evaluation and comparison of the approach by performing experiments on Web-service of two different domains.

The paper is organised as follows. The state of the art is discussed in Section II. In Section III, we present specification of 10 anti-patterns. The concept and architecture of approach used to detect anti-patterns are presented in Section IV. In Section V, the concept of a prototyping tool is discussed. We discuss experimental setup and evaluation of approach in Section VI. The conclusion is presented in Section VII.

II. Stat of the Art

The research on bad smells started in 1999 when Fowler first time introduced 22 code smells and guidelines for refactoring smells. Bad smells are later on discovered at design, architecture and requirement levels. Zhang et al. [20] and Rasool et al. [19] presented reviews on code smells. Bad smells at design levels are called design smells or anti-patterns by different authors. A number of design smell detection techniques and tools are presented by different authors [21, 22, 23, 24, 25]. Anti-patterns and code-smells are often mixed up into one term, the design defects [26]. The code smells are fine-grained and strongly connected to the code-level and anti-patterns are coarse-grained and are shown at the design level. Code smells are code-level symptoms indicating the expected presence of an anti-pattern (also called 'Design Flaw' [27]). Architectural bad smells are also presented by different authors [28, 29, 30]. A review on product line based architectural bad smells is presented by Vale et al. [31]. The concept of bad smells at requirements level and their detection is presented by Femmer et al. [32]. All of the above discussed approaches focused on bad smells for object oriented software applications. The focus of this paper is on bad smells related to service oriented architectures. We discuss in detail the state of the art on bad smells/anti-patterns for service oriented architectures.

A number of books are available on SOA-patterns and principles [8, 33, 34] that provide guidelines and principles characterizing "good" service-oriented designs. These books enable software engineers to manually evaluate the quality of their systems and provide a basis for the enhancement of design and implementation. For example, Rotem-Gal-Oz et-al. [35] suggested 23 SOA-patterns and 4 SOA anti-patterns and they described their effects, causes, and corrections. Erl, in his book [33], presented 80+ SOA design, implementation, security, and governance-related patterns. Kr´al et al. [34] elaborated 7 SOA anti-patterns resulted due to the poor practice of SOA rules. Brown et al. [36] provided the set of 40 anti-patterns. Dudney et al. [37] presented 52 anti-patterns in SOA, and especially in the area of Web-services.

There are few contributions on the identification of patterns from SOA [38, 18, 39]. Upadhyaya et al. [39] presented an approach to detect 9 SOA patterns. Demange et al. [40] presented an approach to detect five SOA patterns from two SOA based systems. It is revealed through the review of literature that the research on Service Oriented Architecture still needs to be explored. Many detection techniques and tools are presented in the literature [21, 23, 25, 26] that focus on specification and detection of OO anti-patterns. These OO based techniques did not give a viable solution for the identification of anti-patterns that are pointed out in web-services. There is a difference between structure of Object Oriented software applications and applications developed using web services. A limited number of approaches are available for the identification of the WS anti-patterns.

Moha et al. [9] presented a technique supplemented with a tool SODA to specify and identify the anti-patterns in SCA systems. Authors performed experiments on two different corpora i.e., Home automation system and Frascati service component architecture. Authors apply algorithms that are not generated manually and they performed experiments on a number of SCA systems to gain the best accuracy. Hence, this approach can only tackle the SCA modules build up using the Java language and are not able to tackle the other SOA technologies like J2EE, SOAP and REST.

Rodriguez et al. [41] described EasySOC and provided a set of guidelines for service providers to avoid bad practices during writing WSDLs. Authors identified eight poor practices that are used to form WSDL template for Web-services. These heuristics are the rules that use pattern matching. A toolset is developed that enforces implementation of guidelines. Authors evaluated effectiveness of the toolset by performing experiments. However, authors did not examine the quality related issues in the web service design.

Coscia et al. [42] presented a statistical correlation analysis on the number of traditional OO metrics and WSDL-level service metrics and found a correlation between them. Anti-patterns in SOAP based web services and REST are introduced first time in [13, 15, 18]. These authors used natural language processing and source code metrics to detect anti-patterns. Anti-patterns of SOAP based web services are detected with high precision and recall but only for some specific services. The tool SODA-W, an extension of SOFA framework [9] uses already established DSL for the detection of SOAP and REST services.

The state of the art approaches discussed above reflects that a large number of authors focused on the detection of anti-patterns from object oriented software projects. We present summarised information about SOA based anti-pattern detection approaches in Table 1. We also realised that SOA based anti-pattern detection approaches focused towards anti-patterns detection for Service oriented architecture specifically for SOAP(Simple Object Access Protocol) based services. We found only three articles on REST APIs anti-patterns detection techniques [15, 18, 19]. The emphasis of above discussed approaches was not on the detection of the anti-patterns in the service interfaces. Sindhgatta et al. [43] presented a comprehensive literature survey on service cohesion, coupling, and reusability metrics, and they come up with five new cohesions and coupling metrics that are set as a new service design requirement.

TABLE I. SUMMARISED INFORMATION ABOUT SOA ANTI-PATTERN DETECTION TECHNIQUES

Reference	Key Concept	Anti-patterns Recovered	Technique	Case Studies	P/R
[9]	A rule-based approach capable for the specification and detection of anti-patterns using a set of metrics.	TS, MS, DS,MS	SOFA	Home-Automation	75%
[12]	SOMAD apply sequential association rules to get execution traces of services.	S, MS, CS, DS, Kt, BS	Association rule mining	Home Automation	90%
[13]	SODA-W is supported by an extended version of SOFA used for specification and detection of SOA anti-patterns from web services.	RP, AN, LCOP, CS, DuS, MRPC, CRUDY-I, GOWS, FGWS	Source Code metrics for static and dynamic analysis	Experiments performed with 13 weather-related and 109 finance related WSs.	75% 100%
[14]	An automated approach for the detection of Web service anti-patterns using a cooperative parallel evolutionary algorithm (P-EA).	MRPC, CRUDYI, DS, AN, FG, GOWS	Parallel Evolutionary Algorithm	Web services from ten different application domains	85to89%
[44]	Genetic Programming approach based on combination of metrics and threshold values	MS, NS, DS, AN	Genetic Programming	310 services of different domains	85% 87%
[45]	Java to WSDL Mapper	EDM, RPT, WET, AN, UFI, IC, ISM, LCOP	Text Mining and meta programming (Java2wsdl)	60 web services	96% 70-74%
[46]	Contract first concept based approach for detecting WSDL based services using EasySOC tool	WSDL based Services	Text Mining, Machine learning and component based software engineering	391 web services	75-80% 78-94%
[47]	Prediction of Web Services Evolution	Ds, MS, NS, CS	ANN algorithm to predict anti-patterns in future releases	5 web services interfaces	81%, 91%
[48]	Identification of Web Service Refactoring Opportunities as a Multi-Objective Problem	MRPC, CRUDYI, DS, AN, FG, GOWS	MOGP(multi-objective genetic programming)	415 web services from 10 different application domains.	94% , 92%
[50]	Comprehensive guidelines along with tool support to enforce these guidelines for the development of web services.	EDM, RPT, WET, AN, UFI, IC, ISM, LCOP	EASY SOC to detect violation of rules in WSDL	A data set of 392 WSDL documents	95.8%
[51]	Correlation analysis between source code metrics and WSDL implementation code	EDM, RPT, WET, AN, UFI, IC, ISM, LCOP	Statistical analysis for detection of anti-patterns	90 different web services	NA
[52]	WSDL document improvement for effective service availability	EDM, RPT, WET, AN, UFI IC, ISM, LCOP	Discoverability and removal of anti-patterns	391 WSDL documents	NA
[53]	Concept of graph model for detection of anti-patterns	GOb	Metrics based approach	Small examples	NA

P (Precision), R (Recall), NA(Not applicable), EDM (Enclosed Data Model), RPT (Redundant Port Type), RDM (Redundant Data Model),WET(What Ever Type) ,AN(Ambiguous names), UFI(Undercover fault Information), IC(Inappropriate Comments), ISM(information within standard messages),LCOP(Low cohesive operations in same port types), MS(Multi Service), NS(Nano Service), DS(Data Service), Kt(the Knot), BNS(Bottle Neck Service), CS(Chatty Service), DuS(Duplicated service),SC(Service Chain), NH(Nobody Home), MRPC(may be Its Not RPC), Gob(God Object)

- It is observed that many techniques have used metrics for the identification of the anti-patterns.
- Different approaches applied source-code parsing techniques to identify the anti-patterns. Source code parsing techniques include the statistical collection of data like counting Lines of Code, measuring Switch Statement Cases and matching or finding other syntax etc. [49].
- The threshold values of metrics are constant in most cases and they are based on one's experience [49].
- A number of approaches in literature did not mention the accuracy of anti-pattern detection [51, 52, 53] that is important for the effectiveness of any approach.
- Based on the above mentioned limitations, we propose unification of metrics-based and parsing based techniques that not only improve the scope in order to identify number of anti-patterns but it also improves accuracy. The required metrics are obtained from the SoaML of Enterprise Architecture, in spite of reinventing the wheel and by examining them directly from the source code.

III. Specification of Anti-Patterns

The specification of web services related anti-patterns is primary step for their accurate detection from web services. The specification of anti-patterns in literature is textual that is hard to use and describe. Due to unavailability of standard specification of web service anti-patterns, we present specification of 10 selected anti-patterns in this section. Our specifications contain detailed information that is important to understand and detect these anti-patterns. The specifications are further used by our approach for the detection of these anti-patterns. We selected these 10 anti-patterns for the specification and detection due to their common existence in different web services.

1) God Object Anti-Pattern
Name: God Object Web-service

Derived from: God Class or Blob in OO Anti-pattern

Short Description: An object that contains all the information related to the whole service and this object also has many methods. This makes its role in the source-code "god-like".

Violated Principle: When an object holds numerous responsibilities

Also known as: "Schizophrenic-class", "divergent-change", "unconnected-responsibilities", "conceptualisation-abuse", "mixed-abstractions" [37].

Variants: "Vague-classes", "abusive-conceptualisation", "non-related data and behaviour", "irrelevant-methods", "discordant-attributes".

Metrics rule:

God object exists if the service contains:

Many methods and has very low cohesion, high response-time and low-availability

where, Many Methods >=10, Cohesion >= 1, High Response-Time >=1

2) Data Web-service Anti-pattern
Name: Data Web-service

Derived from: Data Class in OO anti-patterns

Also known as: "Data class" "record [class]" "no-command classes"

Variants: "Data clumps", "data container"

Short Description: A web-service that performs information retrieval tasks in a distributed environment through accessor operations, like getters and setters.

Violated Principle of Abstraction: This anti-pattern occurs when a class is used as a holder for data without any method of operating on it.

Metrics rule:

Data Web service exists if the service contains: High accessor operations with few parameters and has high cohesion and high primitive parameters

where, Accessor Operations > 50 < 73 and with few parameters and lcom3<=0 and primitive parameters >100

3) Fine Grained WS Anti-pattern
Name: Fine Grained Web-service

Short Description: Fine-grained web-service description regards tiny services out of which the larger ones are composed. That larger one needs to have many coupled web-services. Therefore, it gives rise to higher development complexity, reduced usability. Individual Web-service is less cohesive due to related operations that spread across services of an abstraction.

Violated Principle: This anti-pattern is the result of overdone implementation complexity

Also known as: "Higher-class-complexity"

Variants: "Too much responsibility", "module-mimic"

Metrics rule:

This anti-pattern exists if a service contains: Few operations and has low cohesion and has very high coupling

where, Operations >=1 And <=2, Low Cohesion >=1 And <=2, Coupling >=1 And <=4

4) Ambiguous Name WS Anti-pattern
Name: Ambiguous Name

Short Description: When the developers use the key terms like Port-Types, operation, and message that contains too short and long, or too general terms, or even show the improper use of verbs.

Violated Design Principle: This anti-pattern arises when the class name has a verb only and hold one method with the same name as the class and class has no inheritance

Also known as: "Operation-class", "method turned into class", "single-routine-classes"

Metris Rule:

A service contains: Too long or too short signatures and has too many general terms in operations

where, COUNT [Operations signature length<3 or Operations Signature length > 30] >1 or Ambiguous operations name should have any one (arg, var, obj, foo, param, in, out, str) >1

5) Duplicated Service
Name: Duplicated Web-service

Derived from: Duplicated class in OO anti-pattern following silo approach

Short Description: Duplicated Web-service__contains identical-operations with the similar names and message parameters.

Violated Design Principle: This anti-pattern occurs when two or more abstractions are identical sharing commonalities with their improper use in the design

Metrics Rule:

A web-service having Identical Operations and Identical Port-Types

where, ARIP > 1 And ARIO>1

ARIP= Average Ratio of Identical Port-Types, ARIO= Avg. Ratio of Identical Operations

where, ARIP count all ambiguous names starting from (arg, var, obj, foo)

And ARIO is calculated as all meaningless operations name having length less than 3 and greater than 30.

6) LowCohesiveOperations in the Same Port-Type Anti-pattern
Derived from: Metric Cohesion

Short Description: Many unrelated operations in one port type.
Impacted quality attributes: Flexibility and Effectiveness

Violated Principle: The modularity of a system is composed of a set of cohesive and loosely coupled modules

Metrics Rule:

The service contains: Many methods and has very low cohesion

NOD >=1 and <= 70 And ARAO <=27

ARAO= Average ratio of accessor operations, NOD= Number of operations declared

7) Redundant Port-Types Anti-pattern
Name: Redundant Port-Types

Derived from: Data Replication Enterprise SOA Anti-pattern

Also known as: "Similar Signature Class", "split-identity", "redundant-classes", etc.

Short Description: When a WS contains multitude Port-Types and is composed of a number of redundant operations handling the same messages.

Variants: "Duplicate-design-artifacts"

Violated Principle: This anti-pattern arises when two or more classes have split-identity

Metris Rule:

Web-service contains: Many operations and has many port-types resulting in high cohesion

where, NOPT>1 And NPT >1

NOPT = Num of Operations in Port-Types, NPT= Number of Port-types

8) Chatty Web-service Anti-pattern
Short Description: Chatty-WS is an anti-pattern in which numerous attribute-level operations like getters and setters exist in order to complete an abstraction.

Violated Principle: Violation of coupling and cohesion

Issues: Difficult to infer the order of invocation gives rise to maintenance issues.

Metrics Rule:

Chatty Web-service exists if service contains:

Low Cohesion with High Accessor Methods And Has Low Availability And High Response Time And Many Methods.

Where Low Cohesion >0 And Accessor Methods >=101 And Many Methods >70

9) CRUDy Web-Service Anti-pattern
Derived from: Chatty Interface anti-pattern

Short Description: A web-service design that contains CRUD-type operations, e.g., create (), ready (), delete (), update (). Interfaces designed may have several methods need to be called to accomplish a goal which makes it chatty.

Violated Principle: This web service may violate share only schema and well-defined boundaries tenets that are important for composition of web services

Metric Rule:

A web-service is Chatty if it contains:

Many CRUD-type operations and LOW cohesion and high accessor operations and high procedures LCOM3 <=0 and accessor operations>100 and procedures >70 Crudy Operations>1

Where CRUD-type operations >1

10)Loosey Goosey Web Service Anti-pattern
Name: Loosey Goosey Web Service

Short Description: Services are designed in a complex way that creates problems for further service extensibility and functionality. Services are tightly coupled and not able to answer the user request.

Violated Principle: Tight coupling between service providers and consumers

Metris Rule:

A web service is Loosey goosey if the service interface implementation is single tier and services are loosely coupled And less cohesive

DIT<1 AND AND CBO>1 AND LCOM3>0

Where DIT: Depth of inheritance

IV. Detection Approach

The motivation of our proposed approach stems from our previous work [49] presented for the detection of code smells from different open source software projects. We used the concept of contract first approaches for the implementation of our approach. Contract-first approaches focus on WSDL document to design first and then write that contract by using any programming language. Mostly, web designers and developers prefer this schema as WSDL schema is far richer as compared to the code you designed in any programming language. In different implementation scenarios, XML schema restricts the size of string and can apply different patterns to use contract detail like the use of regular expressions. Moreover, different tools can be used to convert schema file into HTML documentation. The architecture of proposed approach is presented in Figure 1. We input standard definitions of web services discussed in the previous section. The definitions include static and dynamic properties of web services. The static properties include static features of web services such as number of operations, number of port types, number of parameters etc. The dynamic properties include features such as response time and availability. The metric rules are composed based on the static and dynamic properties of web services. The approach applies these metric rules for the detection of a specific anti-pattern. Our detection algorithms use the metrics (i.e. SoA Modelling Language) / SoAML generated by Enterprise Architect. The approach follows three steps process as shown in Figure 1.

Step1

- The interfaces of web services are reverse engineered using JAVA2WSDL[1] tool based on the contract first concept.
- Dynamic properties of web services are measured using SAAJ[2] and SOAP UI[3] tools.

Step 2

- The source code is reverse engineered into an intermediate form using Sparx System Enterprise Architect tool.
- The intermediate representation is used to understand the complete structure of web services and to build queries.

Step 3

- Anti-pattern detection engine is developed based on the static and dynamic properties of each anti-pattern obtained from steps 1&2.

We discuss EA data model, SQL queries and limitations of approach in the following subsections:

- EA Data Model
- Structured Query Language
- Limitations of approach

A. EA Data Model

Our approach depends on Sparx System Enterprise Architect data model that is directly generated from source code. Sparx System Enterprise Architect 11 has the ability to generate data model of different languages directly from the source code. Instead of reinventing the wheel, we relied on the use of metrics i.e., SOA data model to extract relevant features related to any given anti-pattern.

We used SOAP-UI for parsing the code and SQL to extract the required data for the detection of Web-service related anti-patterns from the Service Oriented Modelling Framework of Enterprise Architect. We selected Enterprise Architect tool due to its ability to generate well structured, self-explanatory and detailed (metrics) data model from the source code of 13 programming languages.

B. Structured Query Language

Our approach is based on SQL to extract data from the data model of Sparx System Enterprise Architect. Structured Query Language is very useful database Query language capable of extracting any required data from the Database model. SQL is having enough types and clauses through which we can extract (delete or alter) any required data (if present) from the SQL database. SQL Queries are useful to retrieve huge amount of data and records from database effectively and efficiently. SQL based databases established standards that is adopted by ANSI & ISO. The syntax of SQL commands is simple like English statements.

Examples of SQL commands that we used in our prototype for extracting data from the data model of Enterprise Architect Modelling tool are given below:

1) Cohesion is based on well-known metrics called as LCOM3(Lack of Cohesion among Methods) and calculated as:

LCOM3 = (∑Procedures - (∑Method Accessed / ∑variables)) / (∑procedures - 1);

Procedures= Select count (operationid) from t_operation, object where object. Object_Type='class';

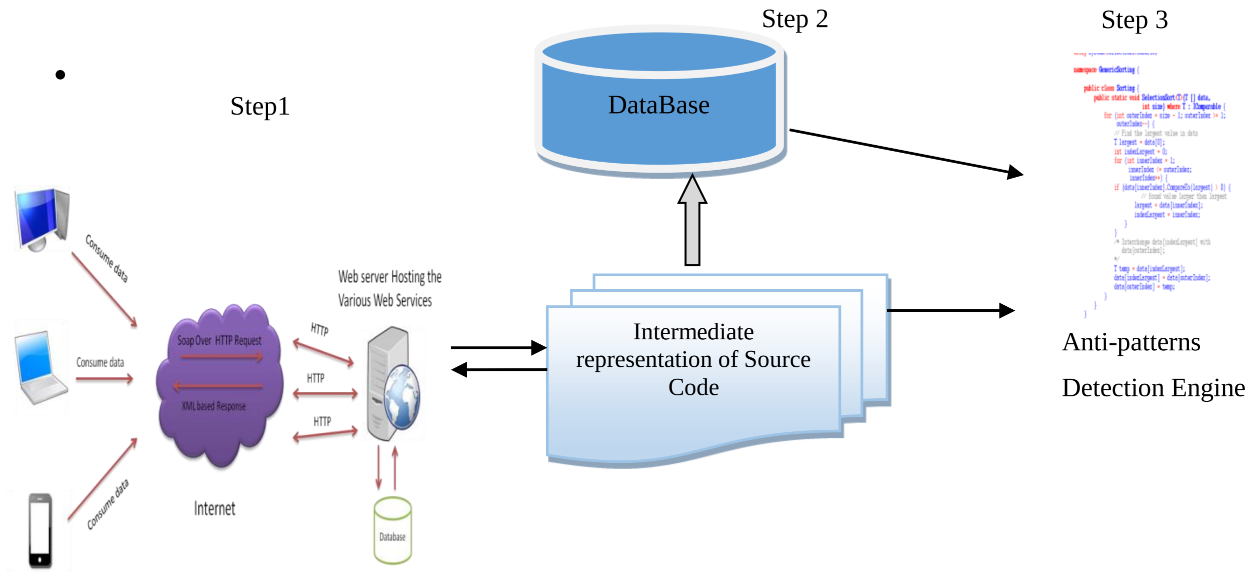
Variables = Select count (operationid) from t_operation, t_object where t_object. Scope='public' and t_object. Object_Type='class';

[1] http://cxf.apache.org/docs/java-to-wsdl.html
[2] http://docs.oracle.com/javaee/5/tutorial/doc/bnbhg.html
[3] https://www.soapui.org/

Method Accessed = Select count (name) from t_operationparams;

2) Coupling is calculated by using CBO (Coupling Between Objects) and is calculated as under:

CBO=∑ coupling among classes/ ∑ classes
Count of Coupling = "select count (connector_id) from t_connector, t_object where t_connector. Connector ID= t_object. Object ID and t_object. Object_Type='class'";



```
namespace GenericSorting {

    public class Sorting {
        public static void SelectionSort<T>(T [] data,
                          int size) where T : IComparable {
            for (int outerIndex = size - 1; outerIndex >= 1;
                 outerIndex--) {
                // Find the largest value in data
                T largest = data[0];
                int indexLargest = 0;
                for (int innerIndex = 1;
                     innerIndex <= outerIndex;
                     innerIndex++) {
                if (data[innerIndex].CompareTo(largest) > 0) {
                          // found value larger than largest
                      largest = data[innerIndex];
                      indexLargest = innerIndex;
                    }
                }
                /* Interchange data[indexLargest] with
                   data[outerIndex];
                */
                T temp = data[indexLargest];
                data[indexLargest] = data[outerIndex];
                data[outerIndex] = temp;
            }
        }
    }
```

Fig. 1. Anti-patterns Detection Approach

Total Classes= select count (object_id) from t_object where Object_Type='class';

3) Primitive types Operations are calculated as:
Select count (operationid) from t_operationparams where type IN('boolean','double','int','byte','short','long','char')");

4) Counting Accessor operations:
Select count (name) from t_operation where name like'set' or name like 'get*'*

5) Cruddy Operations are extracted as:
Select count (t_operation.OperationID) from t_operation, t_object where t_operation.Object_ID = t_object.Object_ID AND t_object.Object_Type = 'Interface' and t_object.Name like 'Create' and t_object.Name like 'update*' and t_object.Name like 'Delete*';*

6) Ambiguous Ports are extracted as:
Select count (object_id) from t_object where name like'arg'or name like 'var*' or name like 'obj*' or name like 'foo*' and object_type='port'*

7) Ambiguous Operations are extracted as:
This metric is based on length of operation name.

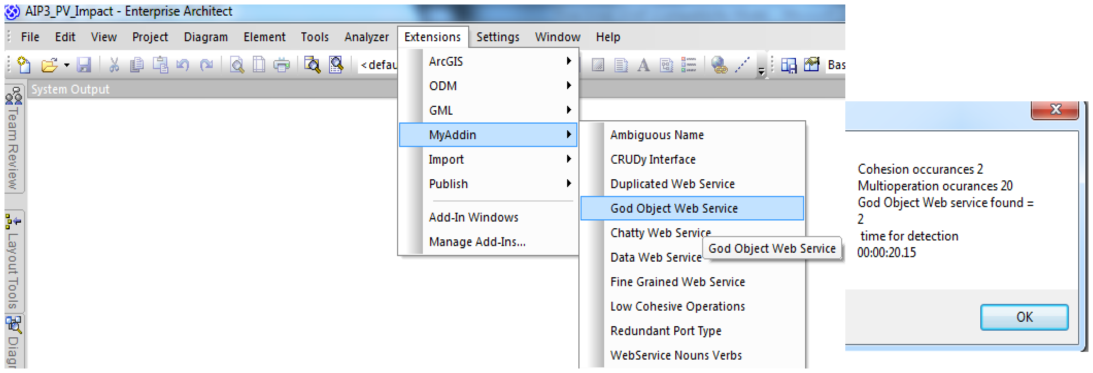
Select count (object_id) from t_object where len(name)<3 or len(name)>30";
Select count (object_id) from t_object where name in ('arg','var*','obj*','foo*' ,'param*','in*','out#','str#');*

C. Limitations of Approach

To detect any given anti-pattern, our approach depends on the metrics i.e., data model of Enterprise Architect. One should have prior knowledge about internal structure of database model created by Sparx System Enterprise Architect to write SQL queries. However, the data model is created only once by reverse engineering source code and it is updatable. A second limitation of our approach is that when we publish contract first then it is harder to change that contract.

V. Prototyping Tool

A prototyping tool is developed to realise concept of approach called Specifying Web-service related anti-patterns and Detection approach named as SWAD. SWAD is an Enterprise Architect plug-in developed using C# language of dot.Net Framework 4.5. The prototype tool is platform independent and it can be integrated with other tools such as IBM Rational Rose, Borland Together and IBM Rhapsody. We selected Enterprise Architect due to our prior experience of using this tool for different other projects [17, 49]. Enterprise Architect has very rich modelling and reverse engineering features for different programming languages. It is easily extendable for multiple languages due to the support of reverse engineering source code of 13+ programming languages. It also generates metrics for the source code written in multiple languages and these metrics are used for the detection of anti-patterns. It directly reverse engineers source code of web services into SOA data model. A screenshot for the user interface of prototyping tool is given in Figure 2.



```
Cohesion occurances 2
Multioperation ocurances 20
God Object Web service found =
2
 time for detection
00:00:20.15
```

Fig. 2. User Interface for detection of Anti-patterns

To demonstrate that SWAD has few distinct features, we compared it with existing state of the art tools SODA-W [13] and P.E.Algo [14]. Table 2 presents a comparison of the different features of SWAD with the two other tools available in the literature. SWAD prototyping tool has a number of features that makes it unique to other two tools. SWAD is scalable and flexible due to the support of Enterprise Architect for generating metrics from various languages.

TABLE II. COMPARISON OF SWAD WITH SODA-W AND P.E. ALGO TOOLS

Features	SWAD	SODA-W[13]	P.E. Algo[14]
Plug-in	Enterprise Architecture	SODA	Eclipse
Extendibility	YES	YES	YES
Platform Independent	YES	YES	YES
Detection-Algorithm Generation	Manual	Manual	Manual to Automatic
Validity for Code – First Web Service	YES	NO	NO
Contract First facility	YES	NO	NO
Number of anti-patterns detected	10	6	7

VI. EVALUATION OF APPROACH

Evaluation of an approach is required to measure its quality, accuracy and effectiveness. To evaluate our approach, we applied SWAD on two distinct sets of WSs i.e., 7 weather related web-services and 60 finance related web-services. These sets of web services are selected due to the availability of their results. We compare our results with two existing state of the art techniques [13, 14] used for detection of anti-patterns from web services. Table 3 shows the statistics of examined web services extracted using CLOC[4] tool available freely on the web.

A. Experimental Results

We selected 60 weather and 7 finance related web services to evaluate our approach and recovered 10 anti-patterns. We selected these datasets due to their free availability and comparison of our results with state of the art approaches.

[4] http://cloc.sourceforge.net/

TABLE III. STATISTICS OF EXAMINED WEB SERVICES

WS	SLOC	Methods	Attributes
BLiquidity	12210	4618	4284
Cloan to Currency	29663	8647	7650
sxBATS	13068	4994	4584
xBondRealTime	26577	6170	4541
Curs	12415	4627	4333
Data	34836	10451	8528
ExchangeRates	13535	5030	4544
MFundService	13530	4930	4527
getImage	16307	5506	5230
Index	11958	4635	4218
Populate	11335	4406	4077
ProhibitedInvestor	11565	4453	4165
StockQuoteService	13331	5383	4923
StockQuotes	19790	6327	5662
sflXML	14941	5771	5079
TaarifCustoms	19678	6565	5919
TaxEconomy	16167	6210	5336
TipoCombio	13268	4959	4608
VerifilterSoap	10500	4204	3878
WebService	11120	4314	4046
wsIndicator	10329	4203	3864
wsStrikon	15078	5588	5217
xCalender	22122	7294	6585
xCharts	32925	5585	3679
xCompensation	18917	6553	5693
xEarningCalender	20340	7137	6374
xEnergy	49670	13952	11433
xEnchanges	21305	7154	6476
xFinance	49377	14458	11503
xFundamentals	23731	7581	6806
xFundata	34821	11384	9405
xFunds	31660	9765	8193
xFuture	58311	1732	766
xGlobalBond	16582	5723	5079
xGlobalFundamentals	21858	6963	6236
xGlobalHistorical	36392	11192	9626
xGlobalRealTime	13829	5273	4741
xIndices	22838	6775	5978
xInsider	35561	11714	9565
xInterbank	77971	7551	4291
xLogos	11385	4611	4257
xMaster	23182	7922	7094
xMetals	57469	16208	12659
xNASDAQ	21183	7059	5846
xNews	15531	5666	5158
xOFAC	16037	5906	5293
xOptions	24044	7896	6520
xOutlook	14899	5353	4937
xReleases	4872	5984	5355

The results of our approach are shown in Tables 4 and 5. Each table presents the names of web-services in first column and then rest of the columns shows anti-patterns with their possible metrics detected.

We selected SODA-W[13] and Parallel Evolutionary Algorithm[14] approaches for comparing results of our approach. SODA-W is a GUI based tool used to detect anti-patterns from SOAP based web services. The detailed information about tool is available at [13]. Parallel Evolutionary Algorithm approach is used to detect anti-patterns for SOAP based services that are based on automatic generated algorithms and threshold values on the metrics.

TABLE IV. RESULTS FOR FINANCE RELATED WEB-SERVICES

Name of Web-Services	**GOWS**	**DWS**	**CWS**	**LCWS**	**FGWS**	**CRUDI**	**RPT**	**Dup-WS**	**ANWS**	**LGWS**	**RT**
BLiquidity	√	√	√	√	√	X	X	√	√	√	1s
Cloan to Currency	√	√	√	√	√	X	X	√	√	√	2s
Finding service	**NR**	NR	NR	NR	NR	X	X	NR	NR	NR	None
xBATS	√	√	√	X	X	X	X	√	√	X	2s
xBondRealTime		√	√	√	√	X	X	√	√	X	None
Curs	√	√	√	√	√	X	X	√	√	X	2s
Data	√	√	√	√	√	X	X	√	√	X	2s
ebsWebTest	NR	NR	NR	NR	NR	X	X	NR	NR	NR	None
ExchangeRates	√	√	√	√	√	X	X	√	√	√	2s
MFundService	√	√	√	√	√	X	X	√	√	√	2s
getImage	√	√	√			X	X	√	√	√	
Index	√	√	√	√	√	X	X	√	√	√	2s
Populate	√	√	√	√	√	X	X	√	√	√	3s
ProhibitedInvestor	√	√	√	√	√	X	X	√	√	√	2s
StockQuoteService	√	√	√	√	√	X	X	√	√	X	3s
StockQuotes	√	√	√	√	√	X	X	√	√	X	2s
sflXML	√	√	√	√	√	X	X	√	√	√	2s
TaarifCustoms	√	√	√	√	√	X	X	√	√	√	4s
TaxEconomy	√	√	√	√	√	X	X	√	√	X	2s
TipoCombio	√	√	√	√	√	X	X	√	√	X	2s
VerifilterSoap	√	√	√	√	√	X	X		√	X	2s
WebService	√	√	√	√	√	X	X	√	√	X	2s
wsIndicator	√	√	√	√	√	X	X	√	√	√	3s
wsStrikon	√	√	√	√	√	X	X	√	√	√	2s
wwwThomas	√	√	√	√	√	X	X		√	√	2s
xAnalyst	√	√	√	√	√	X	X	√	√	√	2s
xBonds	√	√	√	√	√	X	X	√	√	√	2s
xCalender	√	√	√	√	√	X	X	√	√	√	2s
xCharts		√	√	√	√	X	X	√	√	√	2s
xCompensation	√	√	√	√	√	X	X	√	√	√	2s
xCorporateAct	NR	NR	NR	NR	NR	X	X	NR	NR	NR	None
xCorporateActions		NR	√	√	NR	X	X	√	√	√	2s
xCurrency	√	√	√	√	√	X	X	√	√	√	2s
xEarningCalender	√	√	√	√	√	X	X	√	√	√	2s
xEmerging		NR	√	√	NR	X	X	√	√	√	none
xEnergy	√	√	√	√	√	X	X	√	√	X	2s
xEnchanges	√	√	√	√	√	X	X	√	√	X	2s
xFinance	√	√	√	√	√	X	X	√	√	X	2s
xFundamentals	√	√	√	√	√	X	X	√	√	X	2s
xFundata	√	√	√	√	√	X	X	√	√	√	none
xFunds	√	√	√	√	√	X	X	√	√	X	2s
xFuture		√	√	√	√	X	X	√	√	X	2s
xGlobalBond	√	√	√	√	√	X	X	√	√	√	2s
xGlobalFundamentals	√	√	√	√	√	X	X	√	√	√	2s
xGlobalHistorical	√	√	√	√	√	X	X	√	√	√	2s
xGlobalRealTime	√	√	√	√	√	X	X	√	√	√	2s
xIndices		√	√	√	√	X	X	√	√	√	None
xInsider	√	√	√	√	√	X	X	√	√	X	2s
xInterbank		√	√	√	√	X	X	√	√	X	2s
xLogos	√	√	√	√	√	X	X		√	X	2s
xMaster	√	√	√	√	√	X	X	√	√	√	2s
xMetals	√	√	√	√	√	X	X	√	√	√	2s
xMoneyMarket	X	NR	NR	√	NR	X	X		√	√	2s
xNASDAQ	√	√	√	√	√	X	X	√	√	√	2s

xNews	√	√	√	√	√	X	X	√	√	√	2s
xOFAC	√	√	√	√	√	X	X	√	√	√	2s
xOptions	√	√	√	√	√	X	X	√	√	√	None
xOutlook	√	√	√	√	√	X	X	√	√	√	2s
xReleases	√	√	√	√	√	X	X	√	√	√	2s

MO: Multi-Operation Occurrences, **CO**: Cohesion Occurrences, **NPT**: Number of Parameter Type, **NOD**: Number of Operations Declared, **AO**: Accessor Operations, **NOI**: Number of Instances detected, **DT**: Detection Time, **RT**: Response Time, **P**: Precision, **R**: Recall, **NAN**: Num. of Ambiguous names in Port-type, **SLAP**: AmbOp = Ambiguous Operations, **ANA**: Ambiguous names anti-pattern, **NR**: No Response (Service not available)

TABLE V. RESULTS FOR WEATHER-RELATED WEB-SERVICES

Name of Web-services	**GOWS**	**DWS**	**CWS**	**LC WS**	**FGWS**	**CRUDI**	**RPT**	**Dup-WS**	**ANWS**	**LGWS**
AIP3	√	√	√	√	√	√	X	√	√	√
FindingService	X	NR	NR	√	NR	√	X	X	√	√
ndfd	√	√	√	√	√	√	X	√	√	X
soapWS	X	NR	NR	√	NR	NR	X	X	√	X
WeatherForecastService	√	√	√	√	√	X	X	√	√	X
WeatherTerrapin	√	√	√	√	√	X	X	√	√	X
webSky	√	√	√	√	√	X	X	√	√	X

GOWS: Gob Object Web Service, **DWS:** Data Web Service , **CWS**: Cruddy Web Service, **LCWS:** Low Cohesive Web service, **RPT:** Redundant Port Type, **ANWS:** Ambiguous Name Web Service, FGWS: Fine Grained Web service, CRUDY I: Crudy Interface, DupWS:Duplicate Web Service

We combine SQL queries and source code parsing methods and these methods work parallel to detect anti-patterns with better accuracy. The reason for their selection is that SQL queries are easy to customise for recovering anti-patterns with slight variations. Secondly, we have very limited number of approaches available for the identification of web-services related anti-patterns.

TABLE VI. COMPARISON OF RESULTS FOR WEATHER RELATED SERVICES

SWAD Tool			**SODA-W Tool**		
Anti-pattern	**WS**	**Precision**	**Anti-pattern**	**WS**	**Precision**
GWS	Detected	68%	GWS	None detected	----
DWS	Detected	100%	DWS	None detected	----
Chatty WS	Detected	65%	Chatty WS	Detected	50%
LCWS	Detected	95%	LCWS	Detected	100%
FGWS	Detected	98%	FGWS	Detected	100%
DWS	Detected	86%	DWS	None detected	----
ANWS	Detected	93%	ANWS	Detected	100%
CRUDy I	None detected	----	CRUDy I	Detected	50%
RPT	None detected	----	RPT	Detected	100%
MRPC	None detected	----	MRPC	None detected	----

TABLE VII. COMPARISON OF RESULTS FOR FINANCE RELATED WEB-SERVICES

SWAD Tool			**SODA-W Tool**		
Anti-patterns	**WS**	**Precision**	**Anti-pattern**	**WS**	**Precision**
GWS	Detected	42.8%	**GWS**	None detected	----
DWS	Detected	100%	**DWS**	None detected	----
Chatty WS	Detected	42.8%	**Chatty WS**	None detected	----
LCWS	Detected	100%	**LCWS**	Detected	100%
FGWS	Detected	100%	**FGWS**	Detected	66.67%
DWS	Detected	57.1%	**DWS**	None detected	----
ANWS	Detected	100%	**ANWS**	Detected	100%
CRUDy I	None detected	----	**CRUDy I**	None detected	----
RPT	None detected	----	**RPT**	Detected	100%
MRPC	None detected	----	**MRPC**	None detected	----

B. Comparison of Results with P.E Algorithm

Tables 6 and 7 shows the comparison of the detection results of the anti-patterns related to the web-services using Parallel Evolutionary Algorithm (P.E.Algo) and our approach i.e., specifying Web-service related anti-patterns and Detection approach. Both tables listed few web-services on which detection have been performed to assess how efficiently the number of WS-related anti-patterns identified in each given web-service. It can be seen from Table 8 that only one or two WS-related anti-patterns are detected in each web-service. For instance, in the web-service named xOutlook only two anti-patterns have been detected using P.E Algo approach. Similarly, Data Web Service and Cruddy Web Service anti-patterns are detected from xMaster web-service using P.E Algo technique. We can see that our approach is capable of detecting a large number of anti-patterns from different web services as compared to other two state of the art approaches. Figure 3 shows the variation of results by three different approaches on selected web services.

TABLE VIII. COMPARISON OF RESULTS GENERATED BY SWAD

Services/Anti-patterns	**GOWS**			**DWS**			**CWS**			**MNR**			**LCWS**			**RPT**			**ANWS**		
	PE-A	**SODA-W**	**SWAD**	**PE-A**	**SODA-W**	**SWAD**	**PE-A**	**SODA-W**	**SWAD**	**PE-A**	**SODA-W**	**SWAD**	**PE-A**	**SODA-W**	**SWAD**	**PE-A**	**SODA-W**	**SWAD**	**PE-A**	**SODA-W**	**SWAD**
AIP3_PV_Impact	X	X	√	X	X	X	X	X	X	X	X	X	X	X		√	√	X	X	√	√
Finding Service	X	X	X	X	X	X	X	X	X	X	X	X	X	X	√	X	X	√	X	X	√
XBATS	X	X	√	X	X	√	X	X	√	√	X	X	X	X	X	X	X	X	X	X	√
ExchangeRates	X	X	√	X	X	√	X	√	√	X	X	X	X	X	√	X	X	X	√	X	√
xAnalyst	√	X	X	X	X	X	√	X	√	X	X	X	X	X	X	X	X	X	X	X	√
X Master	X	X	√	√	X	√	√	X	√	X	X	X	X	X	√	X	X	X	X	X	√
Xoutlook	X	X	√	X	X	√	X	X	√	X	X	X	X	X	√	X	X	X	√	X	√
Xrelease	X	X	√	X	X	√	√	X	√	X	X	X	X	X	√	X	X	X	X	X	√
Xcompensation	√	X	√	X	X	√	√	X	√	X	X	X	X	X	√	X	X	X	X	X	√

GOWS: Gob Object Web Service, **DWS:** Data Web Service, **CWS**: Cruddy Web Service, **MNR** :May be its not RPC, **LCWS:** Low Cohesive Web service, **RPT:** Redundant Port Type, **ANWS:** Ambiguous Name Web Service

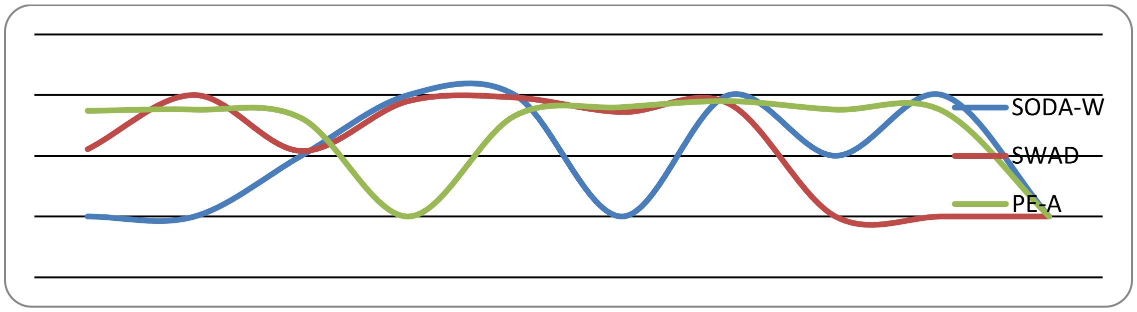


Fig. 3. Variation of Results by three Anti-pattern Detection Tools

VII. CONCLUSION AND FUTURE WORK

The detection of web service anti-patterns from source code supports maintenance, refactoring and highlights poor practices adopted by developers during development of software applications. The detection of anti-patterns from SOA is still young area. A limited number of approaches and tools are presented by different authors for the detection of anti-patterns from SOA based software projects. The state of the art approaches are not flexible for code first and contract first concepts. Our proposed approach has three major contributions. First, we present customisable definitions and algorithms for detection of SOA anti-patterns from multiple languages with varying features. Second, our approach is flexible due to application of SQL queries and regular expressions for matching definitions of anti-patterns in the source code and these searching queries are not hard coded in the source code. Our approach is capable to detect ten SOA anti-patterns from 7 weather related and 60 finance related web services. A prototyping tool is developed to validate the concept of approach. Thirdly, we evaluate our tool on two domains of web services implemented using different programming languages and recovered 10 anti-patterns with improved accuracy. The results of presented approach are compared with two state-of-the-art approaches. The results illustrate the significance of customisable anti-patterns definitions and lightweight searching techniques in order to overcome the accuracy and flexibility issues of previous approaches. We plan to extend our approach for refactoring of recovered anti-patterns. The future work will also focus on detection of anti-patterns from REST APIs.

REFERENCES

[1] Gamma, E., Helm, R., Johnson, R., & Vlissides, J., Design Patterns: Abstraction and Reuse of Object-Oriented Design. European Conference on Object Oriented Programming, 1993.

[2] Riel, A. J. Object-oriented design heuristics (Vol. 335). Reading: Addison Wesley, 1996.

[3] Abbes, M., Khomh, F., Gueheneuc, Y. G., & Antoniol, G. An empirical study of the impact of two antipatterns, blob and spaghetti code, on program comprehension. In *15th European conference on Software maintenance and reengineering (CSMR)*, pp. 181-190, 2012.

[4] Khomh, F., Di Penta, M., Guéhéneuc, Y. G., & Antoniol, G., An exploratory study of the impact of antipatterns on class change-and fault-proneness. *Empirical Software Engineering*, *17*(3), pp. 243-275, 2012.

[5] Harrison, W., & Cook, C., Insights on improving the maintenance process through software measurement. In *Proceedings of Conference on Software Maintenance, 1990*, pp. 37-45, 1990.

[6] Mäntylä, M. V., & Lassenius, C., Subjective evaluation of software evolvability using code smells: An empirical study. *Empirical Software Engineering*, *11*(3), pp. 365-431, 2006.

[7] Arcelli, D., Cortellessa, V., & Trubiani, C., Antipattern-based model refactoring for software performance improvement. In *Proceedings of the 8th international ACM SIGSOFT conference on Quality of Software Architectures* , pp. 33-42, 2012.

[8] E. Thomas, "*Service-Oriented Architecture: Concepts, Technology and Design*," Pearson Education India, 2006.

[9] Palma, F., Nayrolles, M., Moha, N., Guéhéneuc, Y. G., Baudry, B., & Jézéquel, J. M., SOA Antipatterns: An Approach for their Specification and Detection. *International Journal of Cooperative Information Systems*, *22*(4), pp. 1-31, 2013.

[10] Yamashita, A., & Moonen, L., Exploring the impact of inter-smell relations on software maintainability: An empirical study. In *Proceedings of the 2013 International Conference on Software Engineering*, pp. 682-691, 2013.

[11] Liu, H., Ma, Z., Shao, W., & Niu, Z., Schedule of bad smell detection and resolution: A new way to save effort. *IEEE Transactions on Software Engineering*, *38*(1), pp. 220-235, 2012.

[12] Nayrolles, M., Moha, N., & Valtchev, P., Improving SOA antipatterns detection in Service Based Systems by mining execution traces, In Proceedings of WCRE, pp. 321-330. 2013.

[13] Palma, F., Moha, N., Tremblay, G., & Guéhéneuc, Y. G., Specification and detection of soa antipatterns in web services. In *European Conference on Software Architecture*, pp. 58-73, 2014.

[14] Ouni, A., Kessentini, M., Inoue, K., & Cinnéide, M. O., Search-based Web Service Antipatterns Detection, IEEE transaction on services computing, pp. 1-14, 2015.

[15] Palma, F., Gonzalez-Huerta, J., Moha, N., Guéhéneuc, Y. G., & Tremblay, G., Are restful apis well-designed? detection of their linguistic (anti) patterns. In *International Conference on Service-Oriented Computing* , pp. 171-187, 2015.

[16] Petrillo, F., Merle, P., Moha, N., & Guéhéneuc, Y. G., Are REST APIs for Cloud Computing Well-Designed? An Exploratory Study. In *International Conference on Service-Oriented Computing*, pp. 157-170, 2016.

[17] Rasool, G., & Mäder, P., Flexible design pattern detection based on feature types. In Proceedings of 26th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2011, pp. 243-252, 2011.

[18] Palma, F., Dubois, J., Moha, N., & Guéhéneuc, Y. G., Detection of REST patterns and antipatterns: a heuristics-based approach. In *International Conference on Service-Oriented Computing*, pp. 230-244, 2014.

[19] Rasool, G., & Arshad, Z., A review of code smell mining techniques.*Journal of Software: Evolution and Process*, *27*(11), pp. 867-895, 2015.

[20] Zhang, M., Hall, T., & Baddoo, N., Code bad smells: a review of current knowledge. *Journal of Software Maintenance and Evolution: research and practice*, *23*(3), pp. 179-202, 2011.

[21] Kaur, H., & Kaur, P. J., A Study on Detection of Anti-Patterns in Object-Oriented Systems. *International Journal of Computer Applications*,*93*(5), pp. 25-28, 2014.

[22] Erlikh, L., "Leveraging legacy system dollars for E-business". (IEEE) IT Pro, May/June 2000, pp. 17-23.

[23] Moha, N., Guéhéneuc, Y. -G., Duchien, L., Meur, A. -F. L., DECOR: a method for the specification and detection of code and design smells. IEEE Transactions on Software Engineering(2010a), vol. 36, no.1, pp. 20–36, 2010.

[24] Moha, N., Guéhéneuc, Y. -G., Meur, A. -F. L., Duchien, L., Tiberghien, A., From a domain analysis to the specification and detection of code and design smells. Formal Aspects of Computing (FAC), vol. 22, no. 3-4, pp. 345-36, 2010.

[25] Khomh, F., Vaucher, S., Gu´eh´eneuc, Y. -G., Sahraoui, H., Bdtex: A gqm-based bayesian approach for the detection of antipatterns. J. Syst. Softw., vol. 84, no. 4, pp. 559–572, 2011.

[26] Moha, N., Gueheneuc, Y. G., & Leduc, P., Automatic generation of detection algorithms for design defects. In *21st IEEE/ACM International Conference on Automated Software Engineering (ASE'06)*, pp. 297-300, 2006.

[27] Peldszus, S., Kulcsár, G., Lochau, M., & Schulze, S., Continuous detection of design flaws in evolving object-oriented programs using incremental multi-pattern matching. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, pp. 578-589, 2016.

[28] de Andrade, H. S., Almeida, E., & Crnkovic, I., Architectural bad smells in software product lines: An exploratory study. In *Proceedings of the WICSA 2014 Companion Volume* (p. 12), 2014.

[29] Garcia, J., Popescu, D., Edwards, G., & Medvidovic, N., Identifying architectural bad smells. In *13th European Conference on Software Maintenance and Reengineering, CSMR'09*, pp. 255-258, 2009.

[30] Garcia, J., Popescu, D., Edwards, G., & Medvidovic, N., Toward a catalogue of architectural bad smells. In *International Conference on the Quality of Software Architectures*, pp. 146-162, 2009.

[31] Vale, G., Figueiredo, E., Abílio, R., & Costa, H., Bad smells in software product lines: A systematic review. In *Software Components, Architectures and Reuse (SBCARS), 2014 Eighth Brazilian Symposium on*, pp. 84-94, 2014.

[32] Femmer, H., Fernández, D. M., Wagner, S., & Eder, S., Rapid quality assurance with requirements smells. Journal of Systems and Software, 123, 190-213, 2017.

[33] Thomas Erl. Service-Oriented Architecture: Concepts, Technology, and Design. Prentice Hall PTR, August 2005.

[34] Jaroslav Kr´al and Michal Zemliˇcka. Crucial Service-Oriented Antipatterns. volume 2, ˇ pp. 160–171. International Academy, Research and Industry Association (IARIA), 2008.

[35] Rotem-Gal-Oz, E. Bruno and U. Dahan,, "SOA patterns Manning, pp.296, 2012.

[36] W.J Brown, R.C Malveau., H.W. McCormick, T.J Mowbray, "Anti-patterns: Refactoring Software, Architectures, and Projects in Crisis," 1st edn. John Wily and Sons, West Sussex, 1998.

[37] B. Dudney, S.Asbury, J.K. Krozak,.: J2EE AntiPatterns. John Wiley & Sons Inc. August 2003.

[38] D. Penta, Massimiliano, A. Santone, and M. Luisa., Discovery of SOA patterns via model checking. In Proceedings of 2nd international workshop on Service oriented software engineering: in conjunction with the 6th ESEC/FSE joint meeting. ACM, 2007.

[39] Bipin Upadhyaya, Ran Tang, and Ying Zou. An Approach for Mining Service Composition Patterns from Execution Logs. Journal of Software: Evolution and Process, 25(8), pp. 841-870. 2012.

[40] Demange, A., Moha, N., & Tremblay, G., Detection of SOA Patterns, In *International Conference on Service-Oriented Computing* , pp. 114-130, 2013.

[41] Crasso, M., Mateos, C., Zunino, A., & Campo, M., EasySOC: Making web service outsourcing easier. Information Sciences, 259, pp. 452-473, 2014.

[42] Ordiales Coscia, J. L., Mateos, C., Crasso, M., & Zunino, A., Anti-pattern free code-first web services for state-of-the-art Java WSDL generation tools. International Journal of Web and Grid Services, 9(2), 107-126, 2013.

[43] R.Sindhgatta, S.Bikram, and P.Karthikeyan, Measuring the quality of service oriented design., Service-Oriented Computing. Springer Berlin Heidelberg, pp.485-499, 2009.

[44] Ouni, A., Gaikovina Kula, R., Kessentini, M., & Inoue, K., Web service antipatterns detection using genetic programming. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pp. 1351-1358, 2015.

[45] Coscia, J. L. O., Mateos, C., Crasso, M., &Zunino, A., Refactoring code-first Web Services for early avoiding WSDL anti-patterns: Approach and comprehensive assessment. *Science of Computer Programming*, *89*,pp. 374-407, 2014.

[46] Mateos, C., Crasso, M., Zunino, A., & Coscia, J. L. O., Revising WSDL documents: why and how, Part 2. *IEEE Internet Computing*, *17*(5), pp. 46-53, 2013.

[47] Wang, H., Kessentini, M., & Ouni, A., Prediction of Web Services Evolution. In *International Conference on Service-Oriented Computing*, pp. 282-29, 2016.

[48] Wang, H., Ouni, A., Kessentini, M., Maxim, B., & Grosky, W. I., Identification of Web Service Refactoring Opportunities as a Multi-objective Problem. In *2016 IEEE International Conference on Web Services (ICWS)*, pp. 586-593, 2016.

[49] Rasool, G., & Arshad, Z., A Lightweight Approach for Detection of Code Smells. *Arabian Journal for Science and Engineering*, 1-24, 2017.

[50] Rodriguez, J. M., Crasso, M., Mateos, C., & Zunino, A., Best practices for describing, consuming, and discovering web services: a comprehensive toolset. *Software: Practice and Experience*, *43*(6), pp. 613-639, 2013.

[51] Coscia, J. L. O., Mateos, C., Crasso, M., & Zunino, A., Avoiding wsdl bad practices in code-first web services. In *Proceedings of the 12th Argentine Symposium on Software Engineering (ASSE2011)-40th JAIIO* , pp. 1-12, 2011.

[52] Rodriguez, J. M., Crasso, M., Zunino, A., & Campo, M., Improving Web Service descriptions for effective service discovery. *Science of Computer Programming*, *75*(11), pp. 1001-1021, 2010.

[53] YUGOV, A., Approach to anti-pattern detection in service-oriented software systems. Trudy ISP RAN /Proc, 28(2), pp. 79-96, 2016.