Early Quality of Service Prediction via Interface-level Metrics, Code-level Metrics, and Antipatterns

Chaima Abid[a], Marouane Kessentini[a], Hanzhang Wang[b]

[a] *University of Michigan, USA*
[b] *eBay Inc., USA*

Abstract

Context: With the current high trends of deploying and using web services in practice, effective techniques for maintaining high quality of Service are becoming critical for both service providers and subscribers/users. Service providers want to predict the quality of service during early stages of development before releasing them to customers. Service clients consider the quality of service when selecting the best one satisfying their preferences in terms of price/budget and quality between the services offering the same features. The majority of existing studies for the prediction of quality of service are based on clustering algorithms to classify a set of services based on their collected quality attributes. Then, the user can select the best service based on his expectations both in terms of quality and features. However, this assumption requires the deployment of the services before being able to make the prediction and it can be time-consuming to collect the required data of running web services during a period of time. Furthermore, the clustering is only based on well-known quality attributes related to the services performance after deployment. **Objective:** In this paper, we start from the hypothesis that the quality of the source code and interface design can be used as indicators to predict the quality of service attributes without the need to deploy or run the services by the subscribers. **Method:** We collected training data of 707 web services and we used machine learning

Email addresses: `cabid@umich.edu` (Chaima Abid), `marouane@umich.edu` (Marouane Kessentini), `hanzwang@ebay.com` (Hanzhang Wang)

to generate association rules that predict the quality of service based on the interface and code quality metrics, and antipatterns. **Results:** The empirical validation of our prediction techniques shows that the generated association rules have strong support and high confidence which confirms our hypothesis that source code and interface quality metrics/antipatterns are correlated with web service quality attributes which are response time, availability, throughput, successability, reliability, compliance, best practices, latency, and documentation. **Conclusion:** To the best of our knowledge, this paper represents the first study to validate the correlation between interface metrics, source code metrics, antipatterns and quality of service. Another contribution of our work consists of generating association rules between the code/interface metrics and quality of service that can be used for prediction purposes before deploying new releases.

Keywords: Quality of Service, web services, interface metrics, code quality, Performance prediction, anti-patterns.

1. Introduction

Web services are nowadays increasingly used in most of industrial software systems [1, 2, 3]. Thus, it is critical to maintain high quality standards in terms of reliability, reusability, extendability etc. when designing and evolving services such as Google, Amazon, eBay, PayPal, FedEx, etc. The quality of service, related to the code and interface, is important for both the providers and subscribers/users. The providers may want to ensure a high quality of service before releasing them to the users. The users/subscribers prefer to use the service with the best quality of service and reasonable price among those offering the same features. Large-scale web services run on complex systems, spanning multiple data centers and distributed networks, with quality of service depending on diverse factors related to systems, networks, and servers [4]. This dynamic, distributed, and unpredictable nature of the web services infrastructure makes estimating and predicting the quality of service (QoS) metrics challenging, time-consuming, and expensive task.

Several studies have been conducted in the literature to predict the quality of web services based on a set of quality attributes (response time, availability, throughput, successability, reliability, compliance, best practices, latency, and documentation) [5, 6]. The majority of existing work help users selecting the best services based on their preferences and expectations [7, 8, 9, 10]. Clustering algorithms were adapted to classify existing services into multiple preferences then the user can select the cluster of services to investigate based on his preferred quality attributes. Thus, these studies are not actually dedicated to make prediction of services before deployment to potential users so they are not useful for services providers but mainly beneficial for subscribers. Some other studies are related to the prediction of the evolution of web services interface from the history of previous releases' metrics [11]. In another category of work, several approaches have been proposed to detect quality issues such as antipatterns for web services [12, 13, 14]. Antipatterns are defined as commonly occurring design solutions to problems that lead to negative consequences [15]. Ouni et al. [14] defined a set of rules manually based on a combination of quality metrics to identify antipatterns. However, to the best of our knowledge, the problem of predicting the quality of service based on the interface and code quality attributes was not addressed before this paper, which represents the main gap of existing literature.

In this paper, we start from the hypothesis that source code and interface metrics and antipatterns are early indicators for the quality of service (QoS). We focused on the following types of antipatterns: Multi Service, Nano Service, Chatty Service, Data Service and Ambiguous Service. The source code/interface metrics and antipatterns can be used as an early detector of potential QoS issues before the service gets deployed on the cloud. For example, low cohesion of a web service may induce a high response time and a low availability due to the large number of calls between operations at multiple web services that will be generated whenever a request/query is submitted. Another motivation to validate our hypothesis is that service interface attributes, such as the number of port types or messages, can be measured relatively easily compared with measuring the QoS attributes that requires the deployment of the service.

Based on the above hypothesis, we empirically validated that source code and interface level metrics can be used to predict the quality of service (QoS) attributes. Thus, we proposed a novel approach for predicting quality of service by mining interface and code level metrics and antipatterns of 707 services extracted from an existing QoS benchmark [16].

In our approach, we adapted an Apriori clustering algorithm [17] to generate association rules that link source code and interface level metrics with the quality of service. We considered a two-step approach. The first step consists of extracting association rules between interface/code metrics and quality of service attributes. Then, the second step extracts rules that link antipatterns with interface/code/quality metrics. We divided our approach into two steps since the types of antipatterns are limited, not often easy to detect due to their subjectivity, and could vary from one service to the other. We made the dataset that we created to validate all these new hypotheses available in the following link [1] so it can be used by other researchers and practitioners to answer the following research questions :

- **RQ1:** To what extent code/interface quality metrics can predict the QoS attributes?
- **RQ2:** To what extent code/interface can predict the QoS attributes of services with antitpatterns?
- **RQ3:** To what extent the severity of different types of antipattern can be estimated based on their impact on the QoS?

Our contributions are not limited to only a prediction technique but also to validate a new scientific knowledge to the community about the connections between the code/interface/antipatterns and execution of services. The main contributions of this paper can be summarized as follows:

[1]http://kessentini.net/tscdataset.zip

1. We propose an approach to predict the quality of service based on antipatterns and code/interface level quality metrics. our approach is based on understanding the relationships between code/interface metrics and quality of services unlike most of the existing work for QoS prediction which are more based on the clustering of services based on the quality attributes.
2. Our results confirm that several of the antipatterns and code/interface quality metrics are correlated with quality of service attributes based on an extensive empirical validation over 707 web services.
3. We have also identified in our empirical validation the antipatterns that negatively affects QoS attributes the most.

The remainder of this paper is organized as follows: Section 2 is dedicated to background material related to this research. Section 3 surveys relevant related work. Section 4 presents the description of our machine learning approach to identify the association rules between the quality of service metrics and the interface and code quality metrics of web services and antipatterns while Section 5 contains the results of our methodology. Section 5 discusses threats to validity. Finally, we conclude and outline our future research directions in Section 7.

2. Background

In this section, we provide a brief overview of related concepts, including inputs and outputs, used in our study. We will define the different interface/-code metrics, antipatterns and quality of service attributes considered by our approach.

2.1. Quality Metrics Level: Interface, Code and Service

In our work, we identified a set of metrics that can be divided into three categories: interface, code and quality of service attributes. Interface level metrics are used to measure the complexity and the usage of service interfaces (e.g. WSDL files) such as the number of operations. Code level metrics are more related to measure the quality of the source code of the services using mainly static analysis. It is possible for any web service to extract the pseudo code of the implementation of the operations in the interface which is enough to get code level static metrics such as coupling and cohesion.

As Web service technology suggests that the Web service is accessible only through its WSDL, we use the Java^{TM} API for XML Web Services (JAX-WS)[2] to generate the Java artifacts of the Web service including: Depth of Inheritance Tree (DIT), Weighted Methods per Class (WMC), and Coupling Between Objects (CBO). Our approach is based on the *ckjm* tool (Chidamber & Kemerer Java Metrics)[3]. Note that for all code-level metrics were extracted using our parser implemented in our previous work [18].

Table 1 summarizes all the used metrics at different levels.

Table 1: Web service metrics [11]

Metric Name	**Definition**	**Metric Level**
NPT	Number of port types	Interface
NOPT	Average number of operations in port types	Interface
NBS	Number of services	Interface
NIPT	Number of identical port types	Interface
NIOP	Number of identical operations	Interface
ALPS	Average length of port-types signature	Interface
AMTO	Average meaningful terms in operation names	Interface
AMTM	Average meaningful terms in message names	Interface
AMTMP	Average meaningful terms in message parts	Interface
AMTP	Average meaningful terms in port-type names	Interface
NOD	Number of operations declared	Code
NAOD	Number of accessor operations declared	Code
ANIPO	Average number of input parameters in operations	Code

Continued on next page

[2]http://docs.oracle.com/javase/6/docs/technotes/tools/share/wsimport.html
[3]http://gromit.iiar.pwr.wroc.pl/p_inf/ckjm/

Table 1 – continued from previous page

Metric Name	Definition	Metric Level
ANOPO	Average number of output parameters in operations	Code
NOM	Number of messages	Code
NBE	Number of elements of the schemas	Code
NCT	Number of complex types	Code
NST	Number of primitive types	Code
NBB	Number of bindings	Code
NPM	Number of parts per message	Code
COH	Cohesion: The degree of the functional relatedness of the operations of the service	Code
COU	Coupling: A measure of the extent to which inter-dependencies exist between the service modules	Code
ALOS	Average length of operations signature	Code
ALMS	Average length of message signature	Code
Response Time	Time taken to send a request and receive a response	QoS
Availability	How often is the service available for consumption	QoS
Throughput	Total Number of invocations for a given period of time	QoS
Successability	Number of response / number of request messages	QoS
Reliability	Ratio of the number of error messages to total messages	QoS
Compliance	The extent to which a WSDL document follows WSDL specification	QoS
Best Practices	The extent to which a web service follows WS-I Basic Profile	QoS
Latency	Time taken for the server to process a given request	QoS
Documentation	Measure of documentation (i.e. description tags) in WSDL	QoS

2.2. Service Antipaterns

Service antipatterns are examples of recurrent bad design solutions that designers and developers use when implementing a service [15]. They initially appear to be appropriate and effective solutions to a problem, but they end up having bad consequences that outweigh any benefits. Software engineers often introduce antipatterns unintentionally during the initial design or during software development due to bad design decisions, ignorance or time pressure [14]. Antipatterns make the maintenance and the evolution of services hard and time-consuming. Most of these antipatterns can be detected using the interface and code quality metrics that were defined in the previous sub-section [14]. We selected the following types of antipatterns extracted from previous work [14]:

- Multi Service: Also called god object web service, represents a service implementing a multitude of methods related to different business and technical abstractions. This service aggregates too many methods into a single service, and it is not easily reusable because of the low cohesion of its methods and is often unavailable to end-users because it is overloaded [19]
- Nano Service: Is a too fine-grained service whose overhead (communications, maintenance, and so on) outweighs its utility. This antipattern refers to a small web service with few operations implementing only a part of an abstraction. It often requires several coupled web services to complete an abstraction, resulting in higher development complexity, reduced usability [19]
- Chatty Service: Represents an antipattern where a high number of operations, typically attribute-level setters or getters, are required to complete one abstraction. This antipattern may have many fine-grained operations, which degrades the overall performance with higher response time [20, 21]
- Data Service: An antipattern that contains typically accessor operations, i.e., getters and setters. In a distributed environment, some web services may only perform some simple information retrieval or data access operations. A Data web service usually deals with very small messages of primitive types and may have high data cohesion [12]
- Ambiguous Service: Is an antipattern where developers use ambiguous or meaningless names for denoting the main elements of interface elements (e.g., port-types, operations, and messages). Ambiguous names are not semantically and syntactically sound and affect the discoverability and the reusability of a web service [22]

These five antipatterns are the most frequently occurring ones in service based systems based on recent studies [12, 23, 24].

In the next section, we will describe our adaptation of a machine learning algorithm to identify the possible correlations and causalities between the different levels of code/interface quality metrics, antipatterns, and QoS attributes. While several studies have been proposed to detect antipatterns and predict QoS [14, 13, 25, 26, 27], it is not clear if and how code/interface quality metrics and antipatterns may impact the dynamic QoS attributes or what could be the most severe antipatterns on QoS.

3. Related Work

We summarize, in this section, the existing work on studying performance evaluation and QoS prediction of Web services.

Most of the existing studies in the area of the prediction of QoS for web services can be classified in two categories: web services recommendations and web services evolution.

3.1. QoS Prediction for QoS-driven Web services Recommendation

For this category of Web services recommendation, the goal is to predict the unknown QoS values between different service users and different web services, with partially available information, as the result, the optimal web service with the best QoS value can be recommended to the service user for composition [7, 28, 29, 30, 31]. The common approach to recommend web service using QoS prediction is collaborative filtering which includes two main sets of algorithms: Model-based approaches and memory-based approaches [32, 33, 34, 35, 36, 37, 38, 39, 40].

In collaborative filtering, the goal is to calculate the similarities between service users to make prediction for the missing QoS data. Model-based approaches utilize machine learning, pattern recognition and data mining algorithms in order to predict the unknown QoS values. In memory-based collaborative filtering the similarity between users or services is calculated using a user-item rating matrix and then making prediction using a certain algorithm [41].

Shao et al. use [42] collaborative filtering to find the users similarity and predict the unknown QoS of the web services using the available invocation history for similar users. Zhang et al. [43] present another collaborative filtering method to rank the web services using QoS query information. The authors in [44] take an extra step and combine the user-based approach and item-based approach to propose a hybrid collaborative filtering, called WSRec, to predict the QoS in order to recommend a web service based on a computed rank for the QoS values. WSrec has been shown to achieve a good overall prediction accuracy, however it depends on historical QoS data and can suffer from the sparsity problem.

Some other studies [45, 46, 47, 48] predict the quality of web services to recommend web services with acceptable throughput or response time. Zhang et al. [49] propose a fuzzy clustering approach to predict the QoS of a web service in order to make web service recommendation staisfying the user requirements without sacrificing the quality. The work in [50] presents an example of model-based QoS prediction that uses a pattern recognition method. There are also studies focusing on the use of QoS for composing multiple services [8, 9, 10].

Zhu et al. [7] proposed an approach that takes a set of fixed landmarks as references. These references monitor QoS values of all the available web services. The approach clusters all the available services around the references. To predict the QoS value of the users in one cluster, the algorithm uses the QoS information of the similar landmarks in that cluster. The main shortcoming of the collaborative filtering methods is that they heavily depend on the historical web service invocation information. Although, in practice, each user only invokes one or several web services. Therefore, the user-service invocation information is sparse when the number of services is large.

Most of the existing work focus on predicting web service performance based on other consumers' experiences to target the problem of web service recommendation. They use clustering-based approach to predict the quality of service. Their approach is based on the assumption that the consumers, who have similar historical experiences on some services, would have similar experiences on other services which is not always true. They ignore the large heterogeneity among users' views on the QoS. Furthermore, the clustering method presented in their work applies the hard technique that includes the use of a number of computers, known as landmarks, to perform the gathering of the real time QoS data, which is different from our mining technique proposed in this paper.

3.2. Prediction of Web services Evolution

Another category of related work in the area of web services prediction is to predict the evolution of web services. WSDLDiff [51] is a tool that uses structural and textual similarity metrics to detect the changes between different versions of a web services interface. VTracker, a tracking tool suggested in [52], detects changes in WSDL documents using XML differencing techniques. However, these tools are capable of detecting changes between Web Service releases, they do not provide any future changes prediction or recommendation on quality of service interface to the users. In order to address this challenge, [11] proposes a machine learning approach using an Artificial Neural Network to predict the evolution of web services interface from the history of previous release's metric. They utilized these predicted interface metrics to predict and estimate the risk and the quality of the studied web services.

In the area of code quality, there are some studies focusing on antipattern detection in Service-Oriented architecture (SOA) and web services. Rotem-Gal-Oz described the symptoms of a range of SOA antipatterns [53]. Kral et al. [23] listed seven "popular" SOA antipatterns that violate accepted SOA principles. A number of research works have addressed the detection of such antipatterns. Moha et al. [26] have proposed a rule-based approach called SODA for SCA systems (Service Component Architecture). Later, Palma et al. [12] extended this work for Web service antipatterns in SODA-W using declarative rule specification based a domain-specific language (DSL) to specify/identify the key symptoms that characterize an antipattern using a set of WSDL metrics. Rodriguez et al. [54] and Mateos et al. [55] provided a set of guidelines for service providers to avoid bad practices while writing WSDLs based on eight bad practices in the writing of WSDL for web services. Recently, Ouni et al. [14] proposed a search-based approach based on standard GP to find regularities, from examples of web service antipatterns, to be translated into detection rules.

Mateos et al. [56] as an attempt to provide the developers with some metrics as early indications of services interfaces with low quality, low maintainability or high complexity at development time, they have investigated the statistical correlation between complexity/quality and maintainability related WSDL-level service metrics and traditional code-level Object Oriented (OO) quality metrics and they confirmed a significant correlation. In their analysis, for the OO quality metrics they have included Modularity, Adaptability, Reusability, Testability, Portability, and Conformity attributes.

To automate the process of predicting the performance of the web services, Li et al. [57] proposed WebProphet. They extract the dependencies, compute the metrics and then predict the performance. They infer dependencies between web objects by perturbing the download times of individual objects. The shortcoming of this techniques is that, it is time consuming and imprecise.

Tariq et.al. in [58] introduce a tool called What-If Scenario Evaluator (WISE) to predict the response time based on packet traces from web transactions. However the downside of their proposed tool is that they're not taking into account some of the client-side factors affecting the response time experienced by users.

In another study, in order to predict the response time, Chen et al.[59] introduced a new metric, called Link-Gradients to measure the affect of logical link latency on end-to-end response time for distributed applications. However to compute this metric, they assume all the individual changes are independent from each other in the system which can be a correct assumption in smaller application, but not necessarily applicable to more complex web services. They use this metric to predict the response time for untested configuration as well.

To summarize, none of the above studies analyzed the relationships between code/interface metrics/antipatterns and the QoS attributes which is the main contribution of this paper.

4. Approach

In this section, we present an overview of our approach and then we provide details about the algorithm used and how we adapted it for the prediction of the quality of web services.

4.1. Overview

As described in Figure 1, our approach has two main outcomes: 1) the association rules between the code/interface quality metrics and QoS attributes, and 2) the association rules between the Service antipatterns, and code/interface/QoS attributes. Thus, we generate two different predictive models. The outcome of the second predictive model is also important to understand the severity of antipatterns since no prior work investigated it.

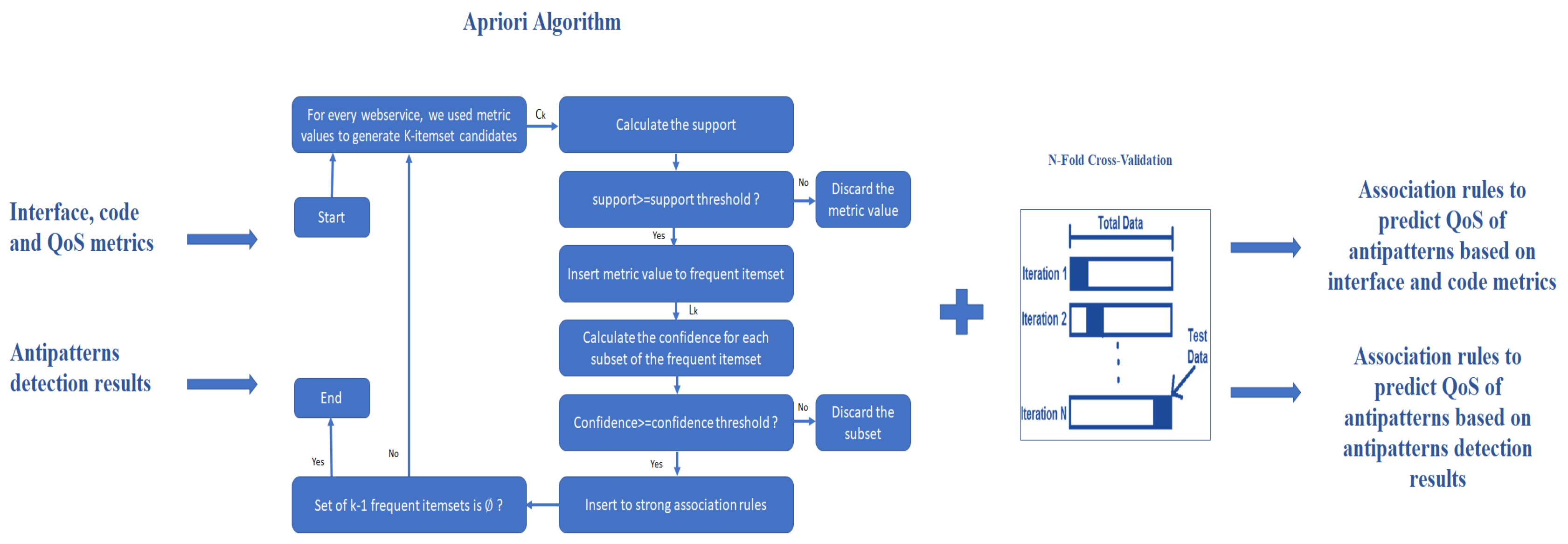

Apriori Algorithm
Interface, code and QoS metrics
Antipatterns detection results
Start
End
For every webservice, we used metric values to generate K-itemset candidates
Ck
Calculate the support
support>=support threshold ?
No
Discard the metric value
Yes
Insert metric value to frequent itemset
Lk
Calculate the confidence for each subset of the frequent itemset
Confidence>=confidence threshold ?
No
Discard the subset
Yes
Insert to strong association rules
Set of k-1 frequent itemsets is ∅ ?
Yes
No
N-Fold Cross-Validation
Total Data
Iteration 1
Iteration 2
Iteration N
Test Data
Association rules to predict QoS of antipatterns based on interface and code metrics
Association rules to predict QoS of antipatterns based on antipatterns detection results

Figure 1: Approach Overview

To generate these outputs, our approach takes as inputs the set of code/interface/QoS metrics calculated on a large data-set of web services along with a list of antipatterns detected on the same data-set using our existing tool [14]. The detection rules used in that tool are described in Table 2. Then, the best association rules are found based on mining the inputs.

Association rules mining is one of the widely studied techniques of data mining [60, 61, 62, 63]. The generated rules represent possible correlations, causality, and redundant patterns between the different dimensions of the analyzed data (e.g. web services quality metrics and antipatterns). In our study, the generated rules take the following template $M \Rightarrow Q$, where M represents either a set of the interface quality metrics or antipatterns, Q is a set of the performance quality attributes (QoS). Therefore, we have $M \cap Q = \emptyset$.

To generate the association rules, the algorithm needs to find, first, the most common itemsets then these patterns will be formalized as a set of rules. The itemset represents the set of our input metrics related to the interface and source code. The frequent itemsets have a high support value which is the percentage of data points in the training data of web services that contain both M and Q. This support value of frequent itemsets should be above the threshold defined as minimum support. Once these frequent itemsets are identified, we adopted the k-fold cross validation method for the association rules generation.

We selected an algorithm called Apriori [17] based on the size and type of data manipulated in our study and the successful application of Apriori to address similar problems [64, 65, 66]. In the next subsections, we present an overview of this algorithm and describe its adaptation to our QoS prediction problem.

4.2. The Apriori Algorithm

The Apriori algorithm was proposed by Agrawal and Srikant in 1994 [17] and has been widely used for frequent itemset mining and association rules learning in databases. The name of the algorithm is Apriori, because it uses the prior knowledge of the frequent itemset properties. It computes the frequent itemsets in the training set through several iterations. Apriori uses the monotonicity property of the support measure to reduce the search space especially with the large data-set of quality metrics and services used in this paper. This rule indicates that all subsets of a frequent itemset must be frequent. Consequently, if an itemset is infrequent then all of its supersets will be infrequent as well. This way, it can eliminate many of the itemsets that are not able to participate in a frequent itemset; and therefore, reduces considerably the running time of the algorithm. There are many versions of Apriori algorithm that improves the performance of association rule mining [67, 61].

The pseudo code for the algorithm is given below for the training set D that consists of the list of metrics, and a support threshold of ε . The Apriori algorithm iteratively find frequent item sets with cardinality from 1 to k (k-itemset). In each iteration, k-frequent item sets are used to find k+1 item sets. For example, we first find the set of frequent 1-itemsets by scanning the dataset, accumulating the count for each item and keeping only those that satisfy minimum support. The results is denoted L1. Next, L1 is used to find L2 the set of frequent 2-itemsets, which is used to find L3, and so on, until no more frequent Then, it uses the frequent item sets to generate association rules. C_k is the candidate set for level k.

Algorithm 1 Pseudo code of the Apriori Algorithm

```
1: Input: A transaction database $D$, and a support threshold of $\epsilon$ .
2: Output: Association rules of support $\geq \epsilon$ .
3: $L_1$= { large1 - itemsets }
4: k= 2
5: **while** $(L_{k-1} \neq \emptyset)$ **do**
6:     $C_k = \{a \cup \{b\} | a \in L_{k-1} \wedge b \notin a\} - \{c | \{s | s \subseteq c \wedge |s| = k-1\} \nsubseteq L_{k-1}\}$
7:     **for** transaction $d \in D$ **do**
8:         $D_t = \{c | c \in C_k \wedge c \subseteq t\}$
9:         **for** candidates $c \in D_t$ **do**
10:            $count[c] = count[c] + 1$
11:        **end for**
11:        $L_k = \{c | c \in C_k \wedge count[c] \leq \epsilon\}$
11:        k=k+1
12:    **end for**
13: **end while**
14: **return** $\bigcup_{i=1}^{\infty} L_i$ =0
```

We describe, in the next sub-section, our adaptation of the Apriori algorithm to our problem.

4.3. Adaptation of the Apriori Algorithm

Figure 1 presents an overview of our adaptation of the Apriori Algorithm. The algorithm is executed twice: a first execution to extract the rules between the code/interface metrics and QoS attributes and a second execution to generate the rules between the antipatterns and QoS attributes. We used Apriori because it is the first proposed algorithm to mine frequent patterns and has been widely used, studied, and is easily accepted. The rules generated by this algorithm are easy to understand and apply. We did not need to use an optimized version of the Apriori because our dataset is relatively small and does not require special computational power or memory. the goal of our paper is to validate the correlations between interface/code metrics and QoS attributes then our plan later is compare which prediction algorithm could be better.

The first execution takes as input an exhaustive list of QoS attributes, presented in Table 1, of a large set of web service releases provided by eBay, Amazon, Yahoo!, etc. and their code/interface quality metrics as detailed later in the experiments section. The output of this step is association rules that predict the performance of web services (QoS).

The second execution of the Apriori learning algorithm takes as input the same data of the first execution along with a list of antipatterns detected on a data-set of web services. The antipatterns are detected using our previous work [14] based on a set of rules presented in Table 2. We selected this detection tool because of the high accuracy and the low false positive as reported in [14]. The output of this step is a set of association rules that can predict the QoS attributes based on the detected antipatterns. This output can be used to understand the severity of different antipattern types on QoS attributes. The two steps of our approach are independent. The goal of the first step is to extract association rules between interface/code metrics and quality of service. The goal of the second step is to generate association rules between antipatterns and quality of service.

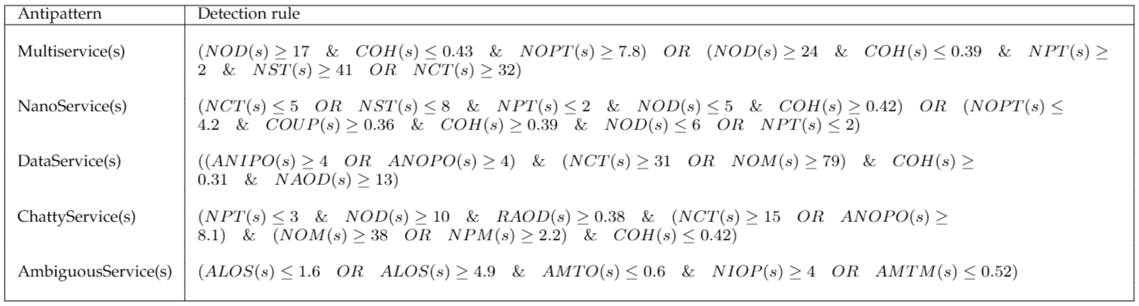
Both executions follow almost the same pattern. We took inspiration from an existing study [68]. In their work, the authors partition the database into two subsets.

As a first step, they choose one of the subsets for training, and leave the other for testing. After that, they mine frequent itemsets from the training subset and use testing subset to compute itemsets' support in whole database. Then, They switch the subsets, so that the previous training set becomes the test set and vice versa. Again, they mine frequent itemsets from training subset and use the testing set to compute supports in whole database. We extended the theorem described in [68] to a more general case by using 5-fold cross-validation based on the number of dimensions (metrics and antipatterns) in the data considered in this paper. Since we have a relatively small dataset size, we used cross-validation as it was proven to be a powerful preventative technique against overfitting [69, 70]. Our approach also guarantees the elimination of the itemsets that are not μ-frequent relative to the whole data set. The training set D is randomly divided into 5 mutually exclusive subsets (the folds) $D_1, D_2, \ldots, D_5$ of approximately equal size where D1 $\cup$ D2 $\cup$ D3 $\cup$ D4 $\cup$ D5 = D. Partitioning the original data in several different ways helps us avoid the possible bias introduced by relying on any one particular partition into test and train components.

After the partitioning step, Apriori algorithm is used to find the set F^{μ}_{D/D_1}, F^{μ}_{D/D_2}, F^{μ}_{D/D_3}, F^{μ}_{D/D_4} and F^{μ}_{D/D_5}. It contains all μ-frequent itemsets relative respectively to D/D_1,D/D_2,D/D_3,D/D_4 and D/D_5. It is possible that some of them are not μ-frequent relative to the whole transaction data set D. Itemsets that are μ-frequent in a subset of the partitions, but not μ-frequent in T are eliminated in the next step.

For every i $\in$ { 1, 2, ..., 5}, We calculate the support of each itemset from F^{μ}_{D/D_i} relative to D. Those itemsets that have $suppcount_D \geq \mu$ are μ-frequent relative to D. They are stored in $F^{\mu}_{D/D_i,D_i}$. The set $F^{\mu}_{D/D_i,D_i}$, contains all μ-frequent itemsets relative to D that appear and are also μ-frequent in D / D_i. We end up with $F^{\mu}_{D/D_1,D_1}$, $F^{\mu}_{D/D_2,D_2}$, $F^{\mu}_{D/D_3,D_3}$, $F^{\mu}_{D/D_4,D_4}$ and $F^{\mu}_{D/D_5,D_5}$ that contain itemsets respectively from F^{μ}_{D/D_1}, F^{μ}_{D/D_2}, F^{μ}_{D/D_3}, F^{μ}_{D/D_4} and F^{μ}_{D/D_5} that are also μ-frequent relative to D. Finally, we obtain the set $F^{\mu}_{D} = F^{\mu}_{D/D_1,D_1} \cup F^{\mu}_{D/D_2,D_2} \cup F^{\mu}_{D/D_3,D_3} \cup F^{\mu}_{D/D_4,D_4} \cup F^{\mu}_{D/D_5,D_5}$ with μ-frequent itemsets in D. Generally, sets $F^{\mu}_{D/D_1,D_1}$, $F^{\mu}_{D/D_2,D_2}$, $F^{\mu}_{D/D_3,D_3}$, $F^{\mu}_{D/D_4,D_4}$ and $F^{\mu}_{D/D_5,D_5}$ are not disjoint.

At the end of our process, we obtain our association rules that predict the QoS from the metrics and the detected quality issues of the web services. The outcome of this research will help both service clients and providers know more about the quality of their web services with the least cost based only on interface and code metrics.



Antipattern	Detection rule
Multiservice(s)	$(NOD(s) \geq 17 \ \& \ COH(s) \leq 0.43 \ \& \ NOPT(s) \geq 7.8) \ OR \ (NOD(s) \geq 24 \ \& \ COH(s) \leq 0.39 \ \& \ NPT(s) \geq 2 \ \& \ NST(s) \geq 41 \ OR \ NCT(s) \geq 32)$
NanoService(s)	$(NCT(s) \leq 5 \ OR \ NST(s) \leq 8 \ \& \ NPT(s) \leq 2 \ \& \ NOD(s) \leq 5 \ \& \ COH(s) \geq 0.42) \ OR \ (NOPT(s) \leq 4.2 \ \& \ COUP(s) \geq 0.36 \ \& \ COH(s) \geq 0.39 \ \& \ NOD(s) \leq 6 \ OR \ NPT(s) \leq 2)$
DataService(s)	$((ANIPO(s) \geq 4 \ OR \ ANOPO(s) \geq 4) \ \& \ (NCT(s) \geq 31 \ OR \ NOM(s) \geq 79) \ \& \ COH(s) \geq 0.31 \ \& \ NAOD(s) \geq 13)$
ChattyService(s)	$(NPT(s) \leq 3 \ \& \ NOD(s) \geq 10 \ \& \ RAOD(s) \geq 0.38 \ \& \ (NCT(s) \geq 15 \ OR \ ANOPO(s) \geq 8.1) \ \& \ (NOM(s) \geq 38 \ OR \ NPM(s) \geq 2.2) \ \& \ COH(s) \leq 0.42)$
AmbiguousService(s)	$(ALOS(s) \leq 1.6 \ OR \ ALOS(s) \geq 4.9 \ \& \ AMTO(s) \leq 0.6 \ \& \ NIOP(s) \geq 4 \ OR \ AMTM(s) \leq 0.52)$

Figure 2: Antipattern Detection rules [14]

To the best of our knowledge, this is the first study aiming to empirically validating the relationships between code/interface quality metrics (or antipatterns) and QoS attributes.

5. Experiment and results

In this section, we cover the data collection and experimental settings. Then, we summarize and discuss the obtained results.

5.1. Data Collection and Evaluation Measures

To answer the different research questions, we built our prediction model for QoS using a large data-set of 707 releases of web services provided by eBay, Amazon, Yahoo!, etc. Besides code and interface metrics, the raw data contains antipatterns detection results of each web service extracted using our previous work as described in the previous section. An important step in generating the association rules is the pre-processing phase for the Apriori algorithm. We did the discretization of the data using a combination of strategies: equal interval width, equal frequency, k-means clustering and categories specifies interval boundaries. We also removed the outliers whenever necessary. To remove the outliers, we performed data visualization (box plot, scatter plot, etc.)and we removed points that are very separate/different from the crowd. Table 2 gives a summary of the considered training set in our experiments that includes a total of 707 services. We selected these 707 active services by contacting each of the web services from that existing benchmark [16] and we found that several of them are not active anymore. Since the existing benchmark is limited to QoS attributes [16], we extended it by calculating the interface/code metrics using a parser that was implemented as part of our previous work [54]. [18]. The new dataset is available at the link [4]. The first file, Dataset1.csv, contains the dataset used to generate association rules linking the code/interface metrics and different quality of service (QoS) attributes. The second file, Dataset2.csv, contains the dataset used to generate association rules linking the code/interface metrics and different quality of service (QoS) attributes for each type of antipattern. It contains code/interface metrics, quality of service (QoS) attributes and the antipatterns detection results. we used the "apriori" function from the "arules" library of R for the apriori algorithm and the "discretize" function from the same library for the discretization. Thus, the experiments are conducted mainly using the R language.

Table 2: Web services used in our dataset

Category	#services	# antipatterns
Financial	107	126
Science	74	104
Search	63	98
Shipping	73	131
Travel	103	154
Weather	53	109
Media	106	214
Education	49	97
Messaging	38	54
Location	41	93
Total	707	1180

To answer **RQ1** and **RQ2**, we validate, first, the proposed approach using a 5-fold cross validation [68], to check if there is significant correlations between the metrics/antipatterns and QoS thus the ability to generate association rules. A small K value for the cross validation means less variance (more bias) while a large K value means more variance (lower bias). We tried different values of k in the cross-validation and we found that k=5 gives the best results in terms of number, meaning and consistency of the rules. The dataset was randomly partitioned into 5 equal size subsamples, again, to avoid any bias. We did take into account the metrics values in the pre-processing

[4]http://kessentini.net/tscdataset.zip

phase for the Apriori algorithm by performing the discretization of data. To this end, we used the following evaluation metrics:

Support: Support is the statistical significance of an association rule interpreting as the ratio (in percentage) of the web services that contain $M1 \cup M2$ (metrics/antipattern types with their thresholds) to the total number of web services in the data-set. Therefore, if that the support of a rule is 5% then it means that 5% of the total web services contain $M1 \cup M2$. In other words,

$$support(M1 \Rightarrow M2) = P(M1 \cup M2) \tag{1}$$

, where P(M1) is the probability of cases containing M1.

Confidence: For a specific number of web services in the data-set, confidence is defined as the ratio of the number of web services that contain $M1 \cup M2$ to the number of web services that contain M1. Thus, if we say that a rule has a confidence of 85%, it means that 85% of the covered web services containing M1 also contain M2. In other words,

$$\begin{aligned} confidence(M1 \Rightarrow M2) &= P(M2|M1) \\ &= \frac{P(XM1 \cup M2)}{P(M1)} \end{aligned} \tag{2}$$

, where P(M1) is the probability of cases containing M1. The confidence of a rule indicates the degree of correlation in the dataset between the different types of metrics/antipatterns and QoS attributes. The Confidence level is considered as a measure to evaluate the strength of the rule. A high confidence is required for the selected association rules.

Lift: Another important measure to evaluate the generated association rules is the lift defined as the confidence of the rule divided by the expected level of confidence. In other words,

$$\begin{aligned} lift(M1 \Rightarrow M2) &= \frac{confidence(M1 \Rightarrow M2)}{P(M2)} \\ &= \frac{P(M1 \cup M2)}{P(M1) * P(M2)} \end{aligned} \tag{3}$$

, where P(M1) is the probability of cases containing M1.

In general, we consider a lift value that is higher than 1 as an indication that the occurrence of M1 has a positive effect on the occurrence of M2 or it confirms that positive correlation between M1 and M2.

If the lift score is smaller than 1, it is considered as an indication that M1 and M2 are not appearing frequently thus the occurrence of M1 has a negative effect on the occurrence of M2 and M1 is negatively correlated with M2. A lift value almost equal to 1 indicates that we cannot conclude about the correlation of M1 and M2.

After validating the correlations to be able to generate statistically significant association rules, we qualitatively validated the rules by identifying the most important interface and code metrics for each of the quality of service QoS attributes. Since this is the first study to generate these association rules, we were not able to compare with any existing studies.

To answer **RQ3**, we evaluated the impact of the 5 different types of antipattern on the QoS attributes by checking the average severity score on each of the QoS attributes. The severity score is defined as the average value of the quality attribute in web services of our data set that did not contain a specific antipattern type divided by the average value of the quality attribute on the web services of our data set containing that antipattern type. We normalized all the quality attributes between 0 and 1 using the min-max normalization (to be minimized). Thus, the highest value is the most severe indication of the impact of an antipattern on each of the quality of service.

5.2. Results

Results for RQ1. Table 5 summarizes the list of the best three association rules linking the code/interface metrics and three different quality of service (QoS) attributes related to response time, reliability and compliance. These rules are obtained by using 5 fold cross validation as described in [68]. We also tried, by trial and error, other values of k for the cross validation but 5 folds gave us the best results. Our model was able to find positive correlations mainly with three out of the eight well-know QoS attributes: response time, availability, throughput, successability, reliability, compliance, latency and documentation. It is expected that not all these quality attributes can be predicted using code and interface level metrics. In fact, some quality attributes such as availability may depend more on hardware requirements but not the quality of the code/interface implementation. Thus, we believe that the results are consistent.

Table 6 contains the average, max and min support, confidence and lift of each rule in table 5. When generating the rules, we used the value 0.6 as a threshold for the support and confidence.

It is clear that the three rules are confirming the strong correlation between response time, compliance and reliability; and many of the quality metrics. For instance, a high response time is correlated with low coupling, an acceptable number of operations per interface (around 12), and high cohesion. Typically, the estimation of these quality of service requires to deploy and run the service then the values will be calculated during a period of time. However, the outcomes of RQ1 confirms that it is possible to predict three of the QoS attributes from the quality of the implementation.

The outcome of the first research question is important for service providers so they can estimate the impact of the quality of their code/interface on the QoS attributes before approving new releases for the users. The generated association rules can be used for reviewing any new pull requests by linking the quality of the code on the QoS attributes.

To summarize, there are strong correlations between three QoS attributes and the quality of the code/interface of services.

Results for RQ2. Table 4 summarizes our findings. All the different five types of antipatterns are strongly correlated with different types of QoS attributes. These rules are obtained using the same fold cross validation to answer RQ1. The table shows the activate rule for each antipattern and the associated QoS attributes. Ambiguous Service antipatterns are experiencing, in general, a high response time which is understandable due to the low reusability and the high coupling in these services. Chatty services and Multi services have the highest negative impact on quality attributes: response time, latency, availability and successability. In fact, these two antipatterns are related to large services including high number of operations and low cohesion which increase the probability of decreasing the quality. Nano service is also correlated based on three different association rules with low best practices and latency due to the small size of these services including few operations.

Table 3 contains the average, max and min support, confidence and lift of each rule in table 4. We also used 0.6 as a threshold for the support and confidence when

Rule ID	average support	max support	min support	average confi-dence	max confi-dence	min confi-dence	average lift	max lift	min lift
4	0.66	0.715	0.6125	0.674493	0.731304	0.626087	1	1.021739	0.98913
5	0.783992	0.922045	0.66232	0.789475	0.922045	0.66232	0.999714	1	0.995068
6	0.726667	0.8	0.633333	0.726667	0.8	0.633333	1	1	1
7	0.811288	0.899159	0.718348	0.893765	0.91536	0.872267	0.998636	1.003929	0.992246
8	0.770913	0.947742	0.613238	0.786026	0.966349	0.625359	1.000092	1.00559	0.997162
9	0.810105	0.831005	0.789204	0.941341	0.96561	0.917072	0.999718	1.000758	0.998679
10	0.815322	0.939007	0.618464	0.840257	0.967647	0.637442	1.001737	1.005185	0.999242

Table 3: Support, confidence and lift of the rules that predict QoS from anti-patterns

generating the rules. The way we read the rules in table 4 is the following: when we know we have one of the antipatterns and one of the left hand sides of the corresponding rules is true, then the right hand side of that rule is also true. For example, when we know we have the antipattern Ambiguous Service and we have AMTO in the range of [0,0.6] then we can conclude that Response Time is in the range of [80.4,368], Successability is within [86.8,100] and Reliability is in the range of [63.8,76.6].

To conclude, we found that the five types of antipatterns have negative impacts on the performance of services and can be used to predict the QoS.

Results for RQ3: Table 4 shows that the most severe antipattern in terms of response time is the Data Service. Among Chatty Service, Data Service, Nano Service and Multi Service, the most severe antipattern in terms of latency is Nano Service. To better investigate the severity of the antipatterns, we compare between the average values of the quality attributes in web services containing a specific type of antipattern comparing to the quality attributes average for the ones without antipatterns. Figure 3 summarizes the severity of antipatterns results. It is clear that the response time quality is the main attribute negatively impacted by most of the antipattern types. Ambiguous services have a high severity on the reliability comparing to the remaining types of antipattern. Chatty services decreased all the quality of service attribute based on the obtained results. Response time, successability and latency are heavily decreased comparing to the remaining quality of service attributes. The results of RQ3 can be used by the service providers to identify the types of antipattern to be fixed

Rule ID	Anti-Pattern	QoS Prediction Rules
4	Ambiguous Service	$AMTO = [0, 0.6) \Rightarrow ResponseTime = [80.4, 368)\ \ \&\ \ Successability = [86.8, 100]\ \ \&\ \ Reliability = [63.8, 76.6)$
5	Chatty Service	$(COH = [0.21, 0.42])\ \ OR\ \ (NOM = [47, 85])\ \ OR\ \ (NCT = [51, 69])\ \ OR\ \ (RAOD = [0.55, 0.74])\ \ OR\ \ (NOD = [23, 42])\ \ OR\ \ (NPT = [1, 3]) \Rightarrow ResponseTime = [55.5, 401)\ \ \&\ \ Latency = [0.33, 58.9)\ \ \&\ \ Compliance = [86.9, 100]\ \ \&\ \ Successability = [87.7, 100]\ \ \&\ \ Reliability = [63, 75.9)$
6	Data Service	$ANOPO = [5.32, 28.5]\ \ OR\ \ NOM = [84, 462]\ \ OR\ \ COH = [0.36, 0.98]\ \ OR\ \ NAOD = [18, 141] \Rightarrow Latency = [1.23, 58.7)\ \ \&\ \ ResponseTime = [227, 1290)\ \ \&\ \ Successability = [91.9, 100]\ \ \&\ \ Documentation = [4, 28.3)\ \ \&\ \ Availability = [90.7, 100]$
7	Multi Service	$NOPT = [7.8, 78]\ \ OR\ \ NCT = [32, 287]\ \ OR\ \ COH = [0.01, 0.43)\ \ OR\ \ NOD = [17, 231] \Rightarrow ResponseTime = [55.5, 574)\ \ \&\ \ Latency = [0.33, 89.7)$
8	Nano Service	$NST = [0, 8) \Rightarrow Successability = [89.6, 100]\ \ \&\ \ BestPractices = [79.6, 93]\ \ \&\ \ Latency = [0.33, 227)\ \ \&\ \ ResponseTime = [46, 635)$
9		$COUP = [0.36, 0.99] \Rightarrow ResponseTime = [46, 635)\ \ \&\ \ Latency = [0.33, 227)$
10		$NPT = [0, 2) \Rightarrow ResponseTime = [46, 635)\ \ \&\ \ Latency = [0.33, 227)\ \ \&\ \ BestPractices = [79.6, 93]$

Table 4: Rules to predict QoS from anti-patterns

based on which quality attribute they want to improve.

In summary, data service, chatty service and multi service are among the severest antipattern types on the quality of service attributes.

6. Threats To Validity

In our experiments, construct validity threats are related to the absence of similar work based on machine learning to predict the QoS. Thus, we were not able to compare our results with any of existing studies. A construct threat can also be related to the fact that we had to manually choose the best discretization method for every metric and train our model based on that.

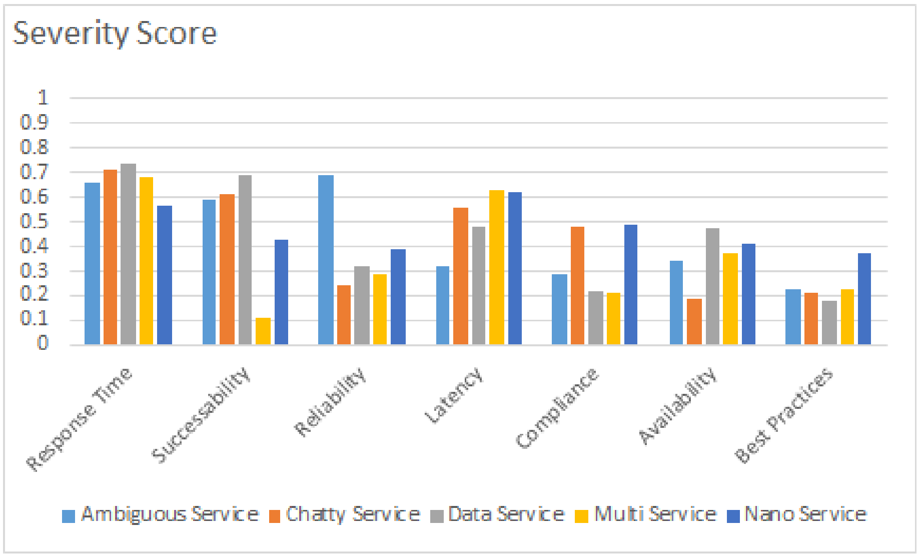


Figure 3: The average severity score of the different types of antipattern on the QoS attributes based on our data set of web services

Internal threats to validity are related to the fact that, in our approach, the prediction is made for each QoS property separately. This isolated prediction is reasonable when the QoS properties are independent, but many QoS properties are correlated, such as response time and latency. The same observation is also valid for the possible combination of multiple antipattern types to predict some quality of service attributes. For instance, multiple instances of both Multi-Service and Chatty-Service can be grouped to predict some quality attributes. We are planning to extend our work to consider such dependencies.

External validity refers to the generalization of our findings. In this study, we performed our experiments on more than 700 web services. A larger dataset is needed to give more reliable results. Since existing studies have confirmed that the programming language affects the quality of the software [71, 72], the impact of antipatterns on the quality of the code might vary from one programming language to another. This can affect the generalization of our results since all Web services considered in our study are written in Java. In our future work, we are planning to include Web services written in other programming languages.

Rule ID	Right hand side of the rule: Performance Metric	Left hand side of the rule: Interface Metrics
1	$ResponseTime = [46, 617)$	$ALMS = [1, 4.24) \quad OR \quad ALOS = [1, 2.77) \quad OR \quad NBB = 1 \quad OR \quad COH = [1.00e-02, 7.46e-01) \quad OR \quad NPT = 1 \quad OR \quad NPM = [0.5, 1.58) \quad OR \quad NBE = [0, 16.1) \quad OR \quad NIOP = [0, 4.62) \quad OR \quad NAOD = [0, 16.2) \quad OR \quad NOPT = [0.33, 12.59) \quad OR \quad ANIPO = [0, 4.75) \quad OR \quad NOM = [2, 29.5)$
2	$Compliance = [86.7, 100]$	$NBB = 1 \quad OR \quad ALMS = [1, 4.24) \quad OR \quad NPT = 1 \quad OR \quad ALOS = [1, 2.77) \quad OR \quad COH = [1.00e-02, 7.46e-01) \quad OR \quad NIOP = [0, 4.62) \quad OR \quad NPM = [0.5, 1.58) \quad OR \quad NAOD = [0, 16.2)$
3	$Reliability = [66.1, 89]$	$NBE = [0, 16.1) \quad OR \quad NOPT = [0.33, 12.59) \quad OR \quad NOM = [2, 29.5) \quad OR \quad NAOD = [0, 16.2) \quad OR \quad NIOP = [0, 4.62) \quad OR \quad NPM = [0.5, 1.58)$

Table 5: Rules to predict QoS

Rule ID	Average Support	Max Support	Min Support	Average Confidence	Max Confidence	Min Confidence	Average Lift	Max Lift	Min Lift
1	0.73063	0.872308	0.647052	0.924195	0.949536	0.901303	1.00808	1.03572	0.98310
2	0.668805	0.713046	0.615475	0.802937	0.875489	0.726074	1.086702	1.18489	0.982685
3	0.694400	0.754661	0.619794	0.861134	0.934675	0.784948	1.13252	1.22924	1.032310

Table 6: Support, confidence and lift of the Rules to predict QoS

7. Conclusion and Future Work

We propose, in this paper, a novel approach to predict QoS with the least cost using code/interface quality metrics and antipatterns. The output of our approach consists of 10 association rules that predict the performance of web services. We used 5 fold cross validation to evaluate the rules. The obtained results based on 707 web services confirm the correlation between both code/interface metrics/antipatterns and the QoS attributes. This important outcome can be used to understand the severity of antipatterns and predict the quality of the services based on the current quality of the implementation.

Our results show that data service, chatty service and multi service are the most severe antipatterns types on the quality of service attributes among the studied antipatterns. All the QMOOD metrics are affected by antipatterns at different levels. Best practices, availability and compliance are the quality metrics deteriorated the most by antipatterns.

As part of our future work, we plan to extend our work to consider other types of antipatterns (such as Redundant PortTypes (RPT), CRUDy Interface (CI) and Maybe It is Not RPC (MNR) [18]) and metrics (such as Performance, Integrity and Usability [6]). Furthermore, we are planning to try other machine learning algorithms such as decision trees for generating association rules and compare their outputs with this work. Finally, we will extend our work to consider the correlation between metrics when predicting the QoS using dimensionality reduction techniques.

References

[1] Pei Guo et al. "Cloud-Based Life Sciences Manufacturing System: Integrated Experiment Management and Data Analysis via Amazon Web Services". In: *INFORMS International Conference on Service Science.* Springer. 2019, pp. 149–159.

[2] Athanasios P Kalogeras et al. "Vertical integration of enterprise industrial systems utilizing web services". In: *IEEE International Workshop on Factory Communication Systems, 2004. Proceedings.* IEEE. 2004, pp. 187–192.

[3] Jieun Jung et al. "Design of smart factory web services based on the industrial internet of things". In: *Proceedings of the 50th Hawaii International Conference on System Sciences.* 2017.

[4] Albert Greenberg et al. "The cost of a cloud: research problems in data center networks". In: *ACM SIGCOMM computer communication review* 39.1 (2008), pp. 68–73.

[5] Aziz Nasridinov, Jeong-Yong Byun, and Young-Ho Park. "A QoS-aware performance prediction for self-healing web service composition". In: *2012 Second International Conference on Cloud and Green Computing.* IEEE. 2012, pp. 799–803.

[6] Ramakanta Mohanty, Vadlamani Ravi, and Manas Ranjan Patra. "Web-services classification using intelligent techniques". In: *Expert Systems with Applications* 37.7 (2010), pp. 5484–5490.

[7] Jieming Zhu et al. “A clustering-based QoS prediction approach for Web service recommendation”. In: *Object/Component/Service-Oriented Real-Time Distributed Computing Workshops (ISORCW), 2012 15th IEEE International Symposium on*. IEEE. 2012, pp. 93–98.

[8] Valeria Cardellini et al. “Flow-based service selection for web service composition supporting multiple qos classes”. In: *Web Services, 2007. ICWS 2007. IEEE International Conference on*. IEEE. 2007, pp. 743–750.

[9] Joyce El Haddad et al. “QoS-driven selection of web services for transactional composition”. In: *2008 IEEE International Conference on Web Services*. IEEE. 2008, pp. 653–660.

[10] Zibin Zheng and Michael R Lyu. “A distributed replication strategy evaluation and selection framework for fault tolerant web services”. In: *Web Services, 2008. ICWS’08. IEEE International Conference on*. IEEE. 2008, pp. 145–152.

[11] Hanzhang Wang, Marouane Kessentini, and Ali Ouni. “Prediction of web services evolution”. In: *International Conference on Service-Oriented Computing*. Springer. 2016, pp. 282–297.

[12] Francis Palma et al. “Specification and detection of SOA antipatterns in web services”. In: *European Conference on Software Architecture*. Springer. 2014, pp. 58–73.

[13] Ali Ouni et al. “Search-based web service antipatterns detection”. In: *IEEE Transactions on Services Computing* 10.4 (2017), pp. 603–617.

[14] Ali Ouni et al. “Web service antipatterns detection using genetic programming”. In: *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*. ACM. 2015, pp. 1351–1358.

[15] William H Brown et al. *AntiPatterns: refactoring software, architectures, and projects in crisis*. John Wiley & Sons, Inc., 1998.

[16] Eyhab Al-Masri and Qusay H Mahmoud. *The qws dataset*. 2008.

[17] Rakesh Agrawal, Ramakrishnan Srikant, et al. “Fast algorithms for mining association rules”. In: *Proc. 20th int. conf. very large data bases, VLDB*. Vol. 1215. 1994, pp. 487–499.

[18] Ali Ouni et al. “Search-based web service antipatterns detection”. In: *IEEE Transactions on Services Computing* 10.4 (2015), pp. 603–617.

[19] Bill Dudney et al. *J2EE antipatterns*. John Wiley & Sons, 2003.

[20] Jose Luis Ordiales Coscia et al. “Anti-pattern free code-first web services for state-of-the-art Java WSDL generation tools”. In: (2013).

[21] Cristian Mateos et al. “Detecting WSDL bad practices in code-first Web Services”. In: *International Journal of Web and Grid Services* 7.4 (2011), p. 357.

[22] José Luis Ordiales Coscia et al. “Refactoring code-first Web Services for early avoiding WSDL anti-patterns: Approach and comprehensive assessment”. In: *Science of Computer Programming* 89 (2014), pp. 374–407.

[23] Jaroslav Král and Michal Žemlicka. “Popular SOA antipatterns”. In: *Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns, 2009. COMPUTATIONWORLD’09. Computation World:* IEEE. 2009, pp. 271–276.

[24] Jaroslav Kral and Michal Zemlicka. “The most important service-oriented antipatterns”. In: *Software Engineering Advances, 2007. ICSEA 2007. International Conference on*. IEEE. 2007, pp. 29–29.

[25] Foutse Khomh et al. “BDTEX: A GQM-based Bayesian approach for the detection of antipatterns”. In: *Journal of Systems and Software* 84.4 (2011), pp. 559–572.

[26] Naouel Moha et al. “Specification and detection of SOA antipatterns”. In: *International Conference on Service-Oriented Computing*. Springer. 2012, pp. 1–16.

[27] Marouane Kessentini et al. “Design defects detection and correction by example”. In: *2011 IEEE 19th International Conference on Program Comprehension*. IEEE. 2011, pp. 81–90.

[28] Marin Silic et al. “Scalable and accurate prediction of availability of atomic web services”. In: *IEEE Transactions on Services Computing* 7.2 (2013), pp. 252–264.

[29] Jieming Zhu et al. “Carp: Context-aware reliability prediction of black-box web services”. In: *2017 IEEE International Conference on Web Services (ICWS)*. IEEE. 2017, pp. 17–24.

[30] Zibin Zheng et al. “Qos-aware web service recommendation by collaborative filtering”. In: *IEEE Transactions on services computing* 4.2 (2010), pp. 140–152.

[31] Marin Silic, Goran Delac, and Sinisa Srbljic. “Prediction of atomic web services reliability for QoS-aware recommendation”. In: *IEEE Transactions on services Computing* 8.3 (2014), pp. 425–438.

[32] Marouane Kessentini, Philip Langer, and Manuel Wimmer. “Searching models, modeling search: On the synergies of SBSE and MDE”. In: *2013 1st International Workshop on Combining Modelling and Search-Based Software Engineering (CMSBSE)*. IEEE. 2013, pp. 51–54.

[33] Marouane Kessentini, Rim Mahaouachi, and Khaled Ghedira. “What you like in design use to correct bad-smells”. In: *Software Quality Journal* 21.4 (2013), pp. 551–571.

[34] Ali Ouni et al. “Prioritizing code-smells correction tasks using chemical reaction optimization”. In: *Software Quality Journal* 23.2 (2015), pp. 323–361.

[35] Boukhdhir Amal et al. “On the use of machine learning and search-based software engineering for ill-defined fitness function: a case study on software refactoring”. In: *International Symposium on Search Based Software Engineering*. Springer, Cham. 2014, pp. 31–45.

[36] Adnane Ghannem, Ghizlane El Boussaidi, and Marouane Kessentini. “On the use of design defect examples to detect model refactoring opportunities”. In: *Software Quality Journal* 24.4 (2016), pp. 947–965.

[37] Usman Mansoor et al. “Multi-view refactoring of class and activity diagrams using a multi-objective evolutionary algorithm”. In: *Software Quality Journal* 25.2 (2017), pp. 473–501.

[38] Hanzhang Wang, Marouane Kessentini, and Ali Ouni. “Bi-level identification of web service defects”. In: *International Conference on Service-Oriented Computing*. Springer, Cham. 2016, pp. 352–368.

[39] Ali Ouni et al. “MORE: A multi-objective refactoring recommendation approach to introducing design patterns and fixing code smells”. In: *Journal of Software: Evolution and Process* 29.5 (2017), e1843.

[40] Martin Fleck et al. “Model transformation modularization as a many-objective optimization problem”. In: *IEEE Transactions on Software Engineering* 43.11 (2017), pp. 1009–1032.

[41] Qi Xie et al. “Personalized context-aware QoS prediction for web services based on collaborative filtering”. In: *International Conference on Advanced Data Mining and Applications*. Springer. 2010, pp. 368–375.

[42] Lingshuang Shao et al. “Personalized qos prediction forweb services via collaborative filtering”. In: *Web Services, 2007. ICWS 2007. IEEE International Conference on*. IEEE. 2007, pp. 439–446.

[43] Qiong Zhang, Chen Ding, and Chi-Hung Chi. “Collaborative filtering based service ranking using invocation histories”. In: *Web Services (ICWS), 2011 IEEE International Conference on*. IEEE. 2011, pp. 195–202.

[44] Zibin Zheng et al. “Wsrec: A collaborative filtering based web service recommender system”. In: *Web Services, 2009. ICWS 2009. IEEE International Conference on*. IEEE. 2009, pp. 437–444.

[45] Lu Li, Mei Rong, and Guangquan Zhang. “A web service qos prediction approach based on multi-dimension qos”. In: *Computer Science & Education (ICCSE), 2011 6th International Conference on*. IEEE. 2011, pp. 1319–1322.

[46] Liang Chen et al. “An enhanced qos prediction approach for service selection”. In: *Services Computing (SCC), 2011 IEEE International Conference on*. IEEE. 2011, pp. 727–728.

[47] Yechun Jiang et al. “An effective web service recommendation method based on personalized collaborative filtering”. In: *2011 IEEE International Conference on Web Services*. IEEE. 2011, pp. 211–218.

[48] Xi Chen et al. “Regionknn: A scalable hybrid collaborative filtering algorithm for personalized web service recommendation”. In: *Web Services (ICWS), 2010 IEEE International Conference on*. IEEE. 2010, pp. 9–16.

[49] Meng Zhang et al. “A web service recommendation approach based on qos prediction using fuzzy clustering”. In: *Services Computing (SCC), 2012 IEEE Ninth International Conference on*. IEEE. 2012, pp. 138–145.

[50] Jike Ge et al. “Web service recommendation based on qos prediction method”. In: *Cognitive Informatics (ICCI), 2010 9th IEEE International Conference on.* IEEE. 2010, pp. 109–112.

[51] Daniele Romano and Martin Pinzger. “Analyzing the evolution of web services using fine-grained changes”. In: *Web Services (ICWS), 2012 IEEE 19th International Conference on.* IEEE. 2012, pp. 392–399.

[52] Marios Fokaefs et al. “An empirical study on web service evolution”. In: *Web Services (ICWS), 2011 IEEE International Conference on.* IEEE. 2011, pp. 49–56.

[53] Juan Manuel Rodriguez et al. “Automatically detecting opportunities for web service descriptions improvement”. In: *Conference on e-Business, e-Services and e-Society.* Springer. 2010, pp. 139–150.

[54] Juan Manuel Rodriguez et al. “Best practices for describing, consuming, and discovering web services: a comprehensive toolset”. In: *Software: Practice and Experience* 43.6 (2013), pp. 613–639.

[55] Cristian Mateos, Juan Manuel Rodriguez, and Alejandro Zunino. “A tool to improve code-first Web services discoverability through text mining techniques”. In: *Software: Practice and Experience* 45.7 (2015), pp. 925–948.

[56] Cristian Mateos et al. “Keeping web service interface complexity low using an oo metric-based early approach”. In: *2016 XLII Latin American Computing Conference (CLEI).* IEEE. 2016, pp. 1–12.

[57] Zhichun Li et al. “WebProphet: Automating Performance Prediction for Web Services.” In: *NSDI.* Vol. 10. 2010, pp. 143–158.

[58] Mukarram Tariq et al. “Answering what-if deployment and configuration questions with wise”. In: *ACM SIGCOMM Computer Communication Review.* Vol. 38. 4. ACM. 2008, pp. 99–110.

[59] Shuyi Chen et al. “Link gradients: Predicting the impact of network latency on multitier applications”. In: *INFOCOM 2009, IEEE.* IEEE. 2009, pp. 2258–2266.

[60] Sergey Brin, Rajeev Motwani, and Craig Silverstein. “Beyond market baskets: Generalizing association rules to correlations”. In: *Acm Sigmod Record* 26.2 (1997), pp. 265–276.

[61] Hannu Toivonen et al. “Sampling large databases for association rules”. In: *VLDB*. Vol. 96. 1996, pp. 134–145.

[62] Mohammed J Zaki et al. “Parallel algorithms for discovery of association rules”. In: *Data mining and knowledge discovery* 1.4 (1997), pp. 343–373.

[63] Wenmin Li Jiawei Han Jian Pei et al. “CMAR: Accurate and efficient classification based on multiple class-association rules”. In: *ICDM-2004* (2001).

[64] M Ilayaraja and T Meyyappan. “Mining medical data to identify frequent diseases using Apriori algorithm”. In: *2013 International Conference on Pattern Recognition, Informatics and Mobile Engineering*. IEEE. 2013, pp. 194–199.

[65] Shiju Sathyadevan, Surya Gangadharan, et al. “Crime analysis and prediction using data mining”. In: *2014 First International Conference on Networks & Soft Computing (ICNSC2014)*. IEEE. 2014, pp. 406–412.

[66] Aditya Methaila et al. “Early heart disease prediction using data mining techniques”. In: *Computer Science & Information Technology Journal* (2014), pp. 53–59.

[67] Ashok Savasere, Edward Robert Omiecinski, and Shamkant B Navathe. *An efficient algorithm for mining association rules in large databases*. Tech. rep. Georgia Institute of Technology, 1995.

[68] Savo Tomović and Predrag Stanišić. “Cross Validation Method in Frequent Itemset Mining”. In: *CECIIS-2011*. 2011.

[69] David F Ransohoff. “Rules of evidence for cancer molecular-marker discovery and validation”. In: *Nature Reviews Cancer* 4.4 (2004), p. 309.

[70] Paul D Adams et al. “Cross-validated maximum likelihood enhances crystallographic simulated annealing refinement”. In: *Proceedings of the National Academy of Sciences* 94.10 (1997), pp. 5018–5023.

[71] Baishakhi Ray et al. “A large scale study of programming languages and code quality in github”. In: *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. 2014, pp. 155–165.

[72] Lutz Prechelt. “An empirical comparison of seven programming languages”. In: *Computer* 33.10 (2000), pp. 23–29.