Dimensionality Reduction of Quality Objectives for Web Services Design Modularization

by

Hussein Skaf

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science (Software Engineering) in the University of Michigan-Dearborn 2018

Master's Thesis Committee:

Associate Professor Marouane Kessentini, Chair
Professor Bruce Maxim
Professor William I. Grosky

DEDICATION

To my Family.

ACKNOWLEDGEMENTS

It is with a great joy that I reserve these few lines of gratitude and deep appreciation to all those who directly or indirectly contributed to the completion of this work:

First and foremost, I offer my sincerest gratitude to Dr. Marouane Kessentini, who dedicated all his wonderful time to collaborate, support and lead me to the end of this piece of work. His advices, dedication, availability, relevant comments, corrections and committeemen led to the success of this work.

I also express my greatest thanks to SBSE members who supported me with valuable feedback and always kindly encouraged me to succeed this project.

I thank all the lecturers of the CIS master degree who have used their valuable time to transmit the knowledge that help in putting this work together me.

Finally, I wish to express my deep gratitude and thank my family who has consistently expressed its unconditional support and encouragement.

All those who contributed in one way or another, to make this work, can be found here, the crowning of their efforts.

Thank you.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

With the increasing use of service-oriented Architecture (SOA) in new software development, there is a growing and urgent need to improve current practice in service-oriented design. To improve the design of Web services, the search for Web services interface modularization solutions deals, in general, with a large set of conflicting quality metrics. Deciding about which and how the quality metrics are used to evaluate generated solutions are always left to the designer. Some of these objectives could be correlated or conflicting.

In this paper, we propose a dimensionality reduction approach based on Non-dominated Sorting Genetic Algorithm (NSGA-II) to address the Web services re-modularization problem. Our approach aims at finding the best-reduced set of objectives (e.g. quality metrics) that can generate near optimal Web services modularization solutions to fix quality issues in Web services interface. The algorithm starts with a large number of interface design quality metrics as objectives (e.g. coupling, cohesion, number of ports, number of port types, and number of antipatterns) that are reduced based on the nonlinear correlation information entropy (NCIE).

The statistical analysis of our results, based on a set of 22 real world Web services provided by Amazon and Yahoo, confirms that our dimensionality reduction Web services interface modularization approach reduced significantly the number of objectives on several case studies to a minimum of 2 objectives and performed significantly better than the state-of-the-art modularization techniques in terms of generating well-designed Web services interface for users.

Keywords - Web service, interface design, modularization, dimensionality reduction approach, NCIE, NSGA-II.

CHAPTER 1: INTRODUCTION

Service-oriented Architecture (SOA) is undeniably an important technology to assist in the construction of complex applications. Indeed, it has effectively contributed to the development of software systems, improving productivity and reducing cost through the reuse of independent and self-contained services. Consequently, many companies are migrating to SOA technology including FedEx[1], Dropbox[2], Google Maps[3], PayPal[4], and so on. Considering the importance and the usefulness of this technology, developing service-oriented systems having a good quality has become paramount. The fundamental principle of SOA is the separation of concerns in which software systems are broken down into smaller services through functional decomposition. Central to the concept was the idea of the application programming interface (API) as the contract to which a service commit [1]. Therefore, a well-designed interface can sharply improve the quality of the whole systems.

However, like any other software systems, Web services, the predominant technology used for implementing SOA, are continuously subjected to change and especially, service interfaces are affected by different types of changes [2] Those changes are necessary to add new functionality according to user's requirements and to fix bugs. Nevertheless, changing Web service becomes increasingly difficult, expensive, and may have a negative impact on the design quality over the time. Continuous change leads to combining many non-cohesive and semantically unrelated operations in the same Web service. In particular, it may weaken the design of the Web service's interface by including a large number of non-cohesive operations. Exposing several operations that are semantically unrelated make the interfaces lack cohesion and on the other side, the service

[1] http://www.fedex.com/caenglish/businesstools/webservices
[2] https://www.dropbox.com/developers/core
[3] developers.google.com/maps/documentation/webservices
[4] developer.paypal.com/webapps/developer/docs/classic/api

suffers from a high degree of interdependence. Therefore, this issue violates the two fundamental concepts of service-oriented design [3] An example of well-known interface design defect is the God Object Web service (GOWS) [4] which implements many operations related to different businesses and technical abstractions in a single service interface leading to low cohesion of its operations and high unavailability to end users because it is overloaded. Indeed, the choice of how operations should be exposed through a service interface can have an impact on the performance, popularity and reusability of the service [5] and it is not a trivial task. On one hand, Web services interface including a high number of operations lead their clients to invoke their interfaces many times which significantly deteriorate the performance. On the other hand, aggregating several operations of an interface into one operation will reduce the reusability of the service. Consequently, there is a high need for efficient re-modularization techniques that we can use to improve the quality of service interfaces.

A variety of work relies on quality of services has been proposed in the literature, and the majority of them proposed to discover the design defects and antipatterns in Web services based on a set of quality metrics related to the interface and code levels [4], [6], [7], [8], [9]. The hard question is how to remove these antipatterns. Despite its importance, service interfaces re-modularization, the main part of the service that needs to be improved, is still in its teenage years [10], [11], [12], [13]. Athanasopoulos et al. represent the first attempt to treat the problem of interface services remodularization [10]. They have formulated the problem as a single-objective problem focusing only on cohesion to find the best decomposition of service interfaces. Ouni et al. suggest that coupling is as important metric as cohesion to drive service interfaces remodularization [11]. However, coupling and cohesion are not enough to generate efficient remodularization and other objectives are also needed to be addressed. In our previous work [12], we selected a high number of objectives to evaluate service interfaces remodularization solutions. Indeed, deciding on how to decompose/modularize an interface is subjective and difficult to automate since a large number of conflicting quality metrics are involved. Thus, multi-objective evolutionary algorithms seem to be efficient to handle a search space of solutions evaluated by conflicting objectives. However, it is known that problems with a high number of objectives cause additional difficulties in terms of the quality of the Pareto set approximation [14] and NSGAII, the one of the most widely used algorithm for generating the Pareto in search-based software engineering (SBSE), is inadequate in our case. Hence, we used dimensionality reduction techniques based on PCA-NSGAII to address the Web services modularization problem. Nevertheless, PCA-NSGAII is a linear dimensionality reduction method and objectives can be reduced are either linearly or nonlinearly correlated, mostly nonlinearly correlated [15].

In this current work, we could go even further in order to select the most representative objectives, we propose a novel dimensionality reduction approach based on nonlinear correlation information entropy (NCIE) [16] embedded in NSGA-II. Our approach aims at finding the best reduced set of objectives (e.g. quality metrics) that can generate near optimal Web services modularization solutions to (i) maximize cohesion, (ii) minimize coupling, (iii) maximize number of port types, (iv) minimize number of operations per port type, and (v) minimize number of antipatterns. The algorithm starts with many quality metrics as objectives that are reduced based on the correlation between them. In the current study, we selected 5 interface design quality metrics coupling, cohesion, number of ports, average of operations per port types, and number of antipatterns.

Therefore, the main contributions of this paper can be summarized as follows: The paper introduces a novel approach of the Web Services re-modularization problem as a reduction approach based on nonlinear correlation information entropy (NCIE) embedded in NSGA-II that takes into account the redundant objectives. We extended our previous work [12] by pointing out the redundant objectives as an important issue. For that, we propose a fully-automated Web services restructuring technique covering these 3 points:

1) **Reduction:** Performing objective reduction on generated population
2) **Evaluation:** Performing non-dominated sorting on selected conflicting objectives
3) **Selection:** Performing crowding distance sorting on selected objectives

We evaluated our approach on two empirical studies. The first one using a set of 22 real-world Web services, provided by Amazon and Yahoo and the second one using 570 Web services, provided by ... Pareto optimality-based algorithm used in this study succeeded in finding a good sequence of re-modularization that should provide the best compromise between used metrics to improve the service interface quality. Statistical analysis of our experiments shows that our dimensionality reduction approach reduced significantly the number of objectives on several case studies to a minimum of 2 objectives and shows that the proposed NCIE outperforms NSGA-II and performed significantly better than the state-of-the-art modularization techniques [10], [11], [12].

The remaining of this paper is organized as follows. Section 2 describes the relevant background and a motivating example for the presented work. Section 3 explains dimensionality reduction approach for Web services modularization. Section 4 provides the different results with existing Web services (Amazon,Yahoo). Section 5 discusses the threats to validity. Section 6 outlines the related work. Finally, section 7 presents the conclusion and future works.

CHAPTER 2: BACKGROUND AND CHALLENGES

To better understand our contribution, we start, in this section, by providing the necessary background of our approach. Then, we discuss a motivating example that illustrates the specific problems that are addressed by our proposal.

2.1 Background

Most of existing real-world Web services interface regroup together high number operations implementing different abstractions such as the Amazon EC2 that contains more than 100 operations in some releases. There are few WSDL design improvement tools [17] that have emerged to provide basic refactorings on WSDL files, however applying these refactorings is fully manual and time consuming. These interface design refactorings correspond to Interface Decomposition, Interface Merging (to merge multiple interfaces) and Move Operation (to move an operation between different interfaces).

2.1.1 Web Service Interface Defects

Web service interface defects are defined as bad design choices that can have a negative impact on the interface quality such as maintainability, changeability and comprehensibility which may impact the usability and popularity of services [18], [19]. They can be also considered as structural characteristics of the interface that may indicate a design problem that makes the service hard to evolve and maintain and trigger refactoring [17]. In fact, most of these defects can emerge during the evolution of a service and represent patterns or aspects of interface design that may cause problems in the further development of the service. To this end, recent studies defined different types of Web services design defects [17], [18], [19]. In our experiments, we considered five types of common Web service antipatterns from the literature [20].

- God object Web service (GOWS): also called Multi service, refers to a service implementing a high number of operations related to different businesses and technical abstractions in a single service.
- Fine grained Web service (FGWS): also called Nano service, is a too fine-grained service, refers to a small Web service whose overhead (communications, maintenance, and so on) outweighs its utility.
- Data Web service (DWS): 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.
- Chatty Web service (CWS): represents an antipattern where a high number of operations are required to complete one abstraction.
- CRUDy Interface (CU): is an antipattern where the design encourages services the RPC-like behavior by declaring create, read, update, and delete (CRUD) operations, e.g., createX(), readY(), etc.

We choose these defect types in our experiments because they are the most frequent and hard to fix [4], [8], [21]. However, the proposed approach in this paper is generic and can be applied to any type of defects.

2.1.2 Used Metrics

We consider five metrics in our approach to deal with the Web service interface remodularization problem. Two metrics are related to the two main properties in restructuring software in general coupling and cohesion. *Coupling* refers to the strength of associations between implementation elements belonging to the same service [22]. We define the coupling metric as the average coupling values between all possible pairs of operations of interfaces. *Cohesion* describes how closely the operations of a service interface are related. We use Lack of Cohesion *(LOC)* metric. Other properties related to the Web service interface level were also suggested: *NPT* metric calculates the number of port-types, *NOPT* metric is the average number of operations per port types, *ANTIs* metric is the number of design antipatterns that can be detected using the rules defined in [4], [6].

2.1.3 Refactoring Types

Our approach supports three high-level refactorings, as described in the following categories:

- Interface Decomposition: Split an interface into multiple port types.
- Interface Consolidation: Merge a set of interfaces that collectively implement a complete, single abstraction into one interface that represents a single cohesive abstraction.
- Move Operation: Move an operation from one interface to another one.

These refactorings are composed by low-level refactorings such as delete and add operations.

2.1.4 Dimensionality Reduction

The remodularization problem is formulating, in general, using optimization problem. Most real-world optimization problems encountered in practice involve multiple criteria to be considered simultaneously. However, in practice, existing evolutionary multi-objective optimization techniques are applied only to problems having smaller number of objectives to find a representative set of Pareto-optimal solutions [23]. Therefore, these methods including NSGA-II may not adequately handle multi-objective optimization problems. Several works have been made to overcome the limitations of existing multi-objective optimization techniques and to make them more effective and efficient [24], [25], [26]. One of the most important proposed work is Objective reduction. The advantages of objective reduction have been highlighted in several studies [27], [28]. The basic goal of this method is to select the smallest set of conflicting objectives by dropping the redundant objectives without changing the Pareto set of the original problem. Removing redundancy and making balance between conflicting objectives are important issues in modeling decision problems.

Conflicting Objectives: Simply, there is conflict between two objectives when good values for one of them imply bad values for another [29]. Brockhoff et al. give the formal definition [28]:

Definition 1: Two objective sets F_1, F_2 are called conflicting if the induced weak Pareto dominance relations differ, that is, $\leq F_1 \neq \leq F_2$ and nonconflicting otherwise ($\leq F_1 = \leq F_2$).

Redundant objectives: One of the objectives can be viewed as a redundant of the other objective if it is not in conflict with it [30]. Brockhoff et al. give the formal definition [28]:

Definition 2: A set F' ⊆ F of objectives is called redundant if and only if there exists an objective F'' ⊂ F' that is nonconflicting with F'.

2.2 Motivating Example

The need to investigate remodularization for service interfaces has been motivated by several issues related to service interfaces design. To illustrate some of these problems, we give a concrete example provided by Amazon. We will use the architecture shown in **figure 1** consists of four Web service interfaces *Amazon Simple Storage Service (AmazonS3), Application Ver1*, *Application Ver7* and *Application Ver11*. *AmazonS3* interface including 16 operations to be used by the other clients. Four operations are invoked by *Application Ver}*, *Application Ver7* invoked 2 other operations and *Application Ver11* invoked 3 other different operations. This solution is, however, undesirable since (1) it implements several features in one service and per consequent, generates an excessive number of calls from different clients to a single interface (2) it implements unrelated operations. This practice violates the design principles and it is critical to fix. To improve this situation, *AmazonS3* can be decomposed into several services that group together the most related operations into a newly created service. Thus, three new services have been introduced *AmazonS3-AccessManage*, *AmazonS3-BucketManage*, and *AmazonS3-ObjectManage*. In this way, we will reduce the number of unrelated operations/functionalities implemented in *AmazonS3*.

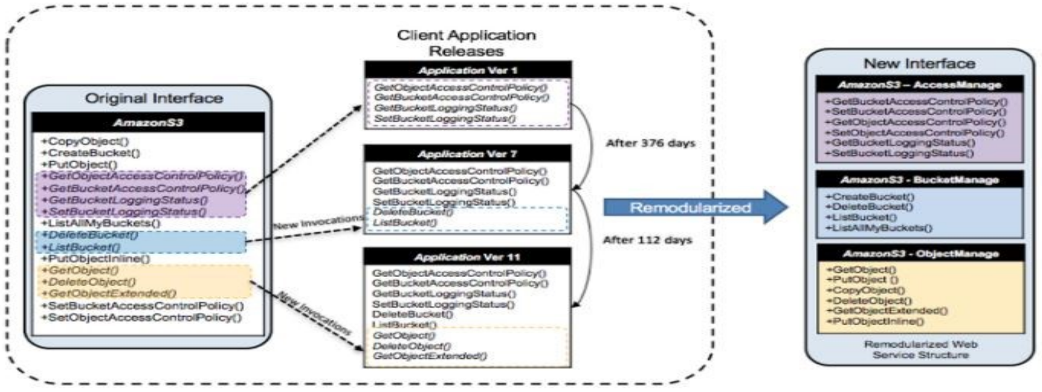


Figure 1: An example of Web service interface modularization

CHAPTER 3: DIMENSIONALITY REDUCTION APPROACH FOR WEB SERVICES MODULARIZATION

We start by outlining the limits of automated refactoring solutions. The first limit that these solutions deal, in general, with a set of conflicting criteria such as several quality metrics or coverage measures. The second limit that some of these objectives could be redundant or correlated, whereas the third one that the current practice is to leave the decision of aggregating fitness functions or treating them separately to the intuition of the SBSE designer.

Wang et al. recently proposed a novel objective reduction approach based on nonlinear correlation information entropy (NCIE) to solve the problems with redundant objectives [30]. Therefore, we used this approach to select the most conflicting objectives during the execution of NSGA-II.

3.1 Approach Overview

The general structure of our approach is sketched in **Figure 2**. The approach takes as inputs a set of quality metrics, several Web services refactoring types, and a Web service to be refactored. The first component consists of a regular execution of NSGA-II during a number of iterations. During this phase, NSGA-II will try to find the non-dominated solutions balancing the initial set containing all the objectives such as improving the quality metrics of the service and minimizing the number of refactorings in the proposed solutions.

After a number of iterations, the second component of the algorithm is executed to analyze the execution traces of the first component (solutions and their evaluations), using the nonlinear correlation information entropy, to check the correlation and the redundancy between the different objectives. This component generates as output the selected objectives. When a redundancy between objectives is detected, we remove the redundant objectives. Then, the first component is executed again with the new objective set.

The whole process of these two components continue until a maximum number of iterations is reached. A set of non-dominated refactoring solutions are proposed to the users with the reduced objectives set to select the best Web service refactorings sequence based on his or her preferences.

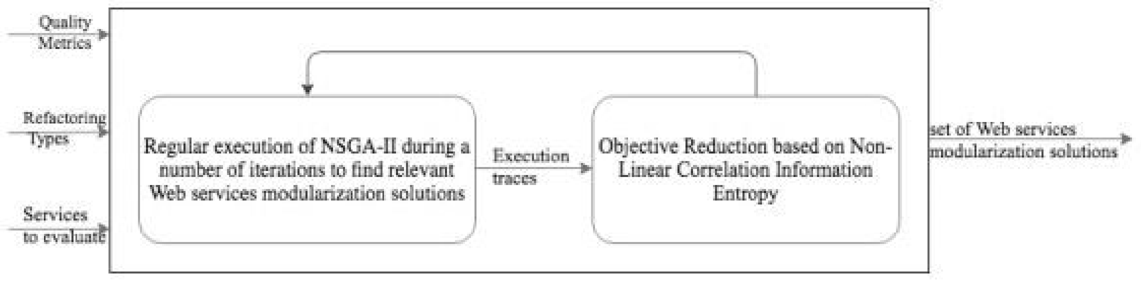

Quality Metrics
Refactoring Types
Services to evaluate
Regular execution of NSGA-II during a number of iterations to find relevant Web services modularization solutions
Execution traces
Objective Reduction based on Non-Linear Correlation Information Entropy
set of Web services modularization solutions

Figure 2: The objective reduction approach embedded in an NSGA-II.

3.2 NSGA-II

NSGA-II is one of the most efficient and popular multi-objective search techniques [16] that has been successfully applied to several engineering optimal design problems [31]. It is based on the genetic algorithm and inspired by the theory of the natural selection in the biological genetics which was developed by Darwin. The basic idea of this algorithm is to make a population of candidate solutions evolve toward the near-optimal trade-off solution in order to solve a multi-objective optimization problem.

As described in Algorithm 1 the first step in NSGA-II is to create randomly an initial population P_0 of individuals encoded using a specific representation (line1). Then, an offspring population Q_0 is generated from the population of parents P_0 (line2) using genetic operators (crossover and mutation). A combined population is formed between both parent and offspring populations into an initial population R_0 of size N (line5). Then, the population R_t is sorted according to Fast-non-dominated-sort (line6). This method is used by NSGA-II to classify a population into non-dominated fronts. In fact, the algorithm is called the Nondominated Sorting Genetic Algorithm, since it is based on a nondominated sorting procedure. The concept of this technique consists of comparing each solution with every other solution in the population to check if it is dominated. The best non-dominated set as selected for the next generation. Fronts are added best ranked first followed by the next-best and so on until the parent population P_{t+1} is filled with N solutions (line8). After that, the solutions of the last selected front are sorted using crowding distance technique (line9). This technique measures how close an individual is to its neighbors. Then, this new population P_{t+1} with N solutions is now used for selection, crossover and mutation in order to create a new population Q_{t+1} with N solutions (Line15). This process is repeated until reaching the maximum number of generations according to stop criteria.

Our approach is applied in every generation of NSGA-II to update the correlation information among objectives. Thus, the next section presents Objective Reduction based on Nonlinear Correlation Information Entropy.

Algorithm 1. High level pseudo code for NSGA-II

```
Begin
Create an initial population $P_0$
Create an offspring population $Q_0$
$t = 0$
**while** stopping criteria not reached **do**
    $R_4 = P_t \cup Q_t$
    F = fast-non-dominated-sort($R_t$)
    $P_{t+1} = \emptyset$ and $i = 1$
    **while** $| P_{t+1} | + | F_i | \leq N$ **do**
        /*Apply crowding-distance-assignment($F_i$)
        $P_{t+1} = P_{t+1} \cup F_i$
        $i = i + 1$
    **end while**
    Sort($F_i \prec n$)
```

```
        $P_{t+1} = P_{t+1} \cup F_i [N - | P_{t+1} |]$
        $Q_{t+1}$ = create-new-population($P_{t+1}$)
        $t = t + 1$
    end while
```

3.3 NSGA-II Adaptation

In this subsection, we describe how we adapted the NSGA-II in terms of solution representation, objective functions and change operators to find the best compromise and the reduced set of objectives (software quality metrics).

3.3.1 Solution Representation

A solution consists of a sequence of n interface change operations assigned to a set of port types. A port type could contain one or many operations but an operation could be assigned to only one port type. A vector-based representation is used to cluster the different operations of the original interface, taken as input from the WSDL file description, into appropriate interfaces, i.e., port types. **Figure 3** describes an example of 5 operations assigned to two port types. As output, a vector representation is automatically translated by our tool into a graphical interface of the modularized Web service.

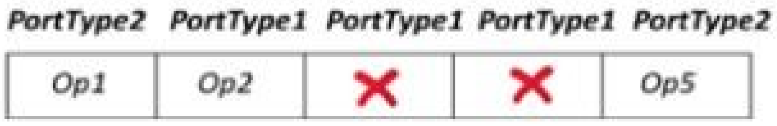


Figure 3: Example of a solution representation

3.3.2 Change Operators

In each search algorithm, the variation operators play the key role of moving within the search space with the aim of driving the search towards optimal solutions. For the crossover, we use the one-point crossover operator. It starts by selecting and splitting at random two parent solutions. Then, this operator creates two child solutions by putting, for the first child, the first part of the first parent with the second part of the second parent, and vice versa for the second child. It is important to note that in multi-objective optimization, it is better to create children that are close to their parents to have a more efficient search process. For mutation, we use the bit-string mutation operator that picks probabilistically one or more refactoring operations from its or their associated sequence and replaces them by other ones from the initial list of possible refactorings. When applying the change operators, different pre- and post-conditions are checked to ensure the applicability of the newly generated solutions such as removing redundant operations or conflicts between operations such as assigning the same operation to two different port types.

3.3.3 Objective Functions and Solution Evaluation

Once the solution has been created, we should formalize the evaluation of an individual as a fitness function. For that, we used the 5 quality attributes along with the number of refactorings in the solutions (solution size) as fitness functions of our algorithm. In our approach we optimize the following objectives:

1) **Coupling:** This objective measures the overall coupling among interfaces in a modularization *M*. It is defined as follows:

$$Coupling(\mathcal{M}) = \frac{\sum_{\substack{\forall (si_i, si_j) \in \mathcal{M} \\ si_i \neq si_j}} Cpl(si_i, si_j)}{\frac{|\mathcal{M}| \times (|\mathcal{M}|-1)}{2}} \qquad (1)$$

where *$Cpl(si_i, si_j)$* was defined in [11].

2) **Cohesion:** The cohesion objective function corresponds to the average cohesion score of each port type in a Modularization M and it is calculated as follows:

$$Cohesion(\mathcal{M}) = 1 - \frac{\sum_{\forall si \in \mathcal{M}} LoC(si)}{|\mathcal{M}|} \qquad (2)$$

where *$LoC(si_i)$* denotes the total interface lack of cohesion, and | M | is the total number of interfaces in the modularization M.

3) **Number of port types:** This objective expected to be maximized to avoid having all operations in one port types. It's defined as follows:

$$NPT(\mathcal{M}) = |\mathcal{M}| \qquad (3)$$

4) **Number of operations per port types:** This objective is ought to be minimized to avoid the unrelated operations in the same port types. It's measured as follows:

$$NOPT(\mathcal{M}) = \frac{\sum_{\forall si \in \mathcal{M}} Size(si)}{|\mathcal{M}|} \qquad (4)$$

where *Size(si)* returns the number of operations in the interface *si*.

5) **Number of antipatterns:** Antipatterns have negative impact on the interface quality and thus, this objective is expected to be minimized. It is calculated as follows:

$$ANTIs(\mathcal{M}) = \sum_{\forall si \in \mathcal{M}} NbAntis(si) \qquad (5)$$

where *NbAntis(si)* returns the total number of detected defect types, introduced previously, in the interface *si*.

The initial iterations of NSGA-II will use all the 5 fitness functions as input then the algorithm will reduce the number of objectives by mining the execution traces (the solutions and their evaluations).

3.4 Objective Reduction based on Nonlinear Correlation Information Entropy

The general structure of this approach is defined in 3 steps: 1) Measuring the correlation among objectives; 2) Classifying objectives by separating the objectives correlated from those non-correlated to a given objective; 3) Selecting the most conflicting objectives. The following subsections give more details about these steps.

3.4.1 Correlation Analysis

The first key step of our approach is to measure the relationship of objectives. A nonlinear correlation information entropy (NCIE) can be used as a robust correlation metric. The main difference between NCIE and the other existing correlation analysis approaches is that NCIE can handle both linear and non-linear correlation [30]. In this study, we use the modified NCIE matrix to measure the redundant and conflicting objectives. NCIE is modified by adding the information of covariance. The sign of the covariance therefore shows whether two objectives are in conflict. The modified NCIE is shown in Eq. 6

$$R^N = \{Sgn(Cov_{ij})NCC_{ij}\}, (1 \leq i, j \leq k) \qquad (6)$$

where cov_{ij} is the ij*th* element in the correlation matrix.

NCC_{ij} denotes the nonlinear correlation coefficient of the i*th* and j*th* variable [32]. It quantizes the correlation in [0,1] for linear and nonlinear cases. If the result is a large positive value, the two objectives are highly positively correlated; if it is a large negative value, the two objectives are highly conflicted and if the result is around zero, the two objectives are not correlated.

3.4.2 Classification for correlated and non-correlated objectives

The basic idea is to separate the objective correlated to objective f_i from those non-correlated to objective f_i [30] by determining the correlation degree.

Algorithm 2. Pseudo code of classifying objectives.

Input: R-modified NCIE of the remaining objectives to objective f_i ($R = R^N(i, j)$, ($f_j \in S_t$) and ($f_i \in S_r$)), m-number of objectives.

Output: R' with classification for correlated and non-correlated objectives.

Add values 0 and 1 (the boundary of non-correlated and correlated) to R^N as R^*.

Sort R^* to R' in an ascending order.

Classify R' into two clusters (R' [$1 : k$] and R' [$k + 1$: end]).

In other words, cut across the least dense area between the two clusters.

R' [$1 : k$] without 0 is the cluster of objectives non-correlated to f_i [$k + 1$: *end*]) without 1 is the cluster of objectives correlated to f_i.

Algorithm 2 takes as input modified NCIE matrix and the number of objectives and returns as output a new matrix with classification for correlated and non-correlated objectives. The situation that all the objectives are non-correlated or correlated to f_i is handled by adding 0 and 1 values (line 3). Then, the matrix is sorting in an ascending order (line4). Afterward, k-means clustering has been used to cut across the least dense area between the two clusters (line5). At the end, 2 clusters are obtained, the first is the cluster of objectives non-correlated to f_i and the second is the cluster of objectives correlated to f_i (line6).

3.4.3 Objective Selection

Objective selection is another key step in the objective reduction approach. According to the obtained NCIE matrix from correlation analysis, we select the most conflicting objectives.

Algorithm 3 takes as input modified NCIE matrix of the current population and the number of objectives and returns as output the selected objective set S_r. The T-threshold is calculated based on classification (Line3). Since classification is separating the objectives into 2 clusters, the threshold is set based on the values in the clusters (Line4). Then, selecting the most representative and the most conflicting objectives, which is the objective with the largest absolute sum of its NCIEs to other objectives (Line [5-10]).

The selected conflicting objectives are stored in to be after outputted (Line11). The objectives that are positively correlated with the stored objectives are omitted (Line [12-13]).

Algorithm 3. Pseudo code of the objective selection.

Input: R-modified NCIE matrix of the current population, m-number of objectives.

Output: S_r-selected objective set

Parameter: T-threshold

```
$S_t = [1 : m]$, $S_r = \emptyset$.
while $S_t \neq \emptyset$.
    if all the elements in $R^N$ are positive, then
        $J = argmax(sum(R^N (1 : m, j)))$. Find the most representative objective
    else
        $J = argmi(sum(R^N (i, j)))$, where $R^N (i, j) < 0$ and $1 \leq i \leq m$.
        Find the most conflicting objective with remaining objectives
    end if
    Move $f_J$ from $S_t$ to $S_r$.
    $F = \{f_J | R^N (J, j) > T_{max}(R^N (J, j)) (f_J \in S_t)\}$. Find set F with the objectives correlated to $f_J$.
    Delete set F from $S_t$.
end while
```

CHAPTER 4: VALIDATION

To evaluate the efficiency of our reduction approach based on nonlinear correlation information entropy for improving the quality of service interfaces, we conducted experiments based on a set of 22 real-world Web services. The obtained results are statistically analyzed with the aim of comparing our proposal with a variety of existing state-of-the-art modularization approaches. In this section, we first present our research questions and then describe the experimental setup and finally discuss the obtained results.

4.1 Research Questions

To evaluate the relevance and performance of the proposed dimensionality reduction approach, we defined the following four research questions:

- **RQ1**: To what extent can the proposed dimensionality reduction approach recommends useful Web service refactorings?
- **RQ2**: To what extent does the proposed dimensionality reduction approach reduce the number of objectives while recommending useful refactorings?
- **RQ3**: How does the proposed dimensionality reduction approach perform compared to other existing Web services modularization techniques not based on computational search [10], [11] and our previous work [12]?
- **RQ4**: What is the impact of the suggested remodularizations by our approach on design quality metrics?

To answer **RQ1**, we considered both automatic and manual validations to evaluate the usefulness of the proposed Web service refactorings. For the manual validation, we asked groups of potential users of our tool to manually evaluate whether the suggested refactorings are feasible and efficient at improving the services quality and achieving their maintainability objectives. We define the metric Manual Correctness (MC) that corresponds to the number of meaningful refactorings divided by the total number of suggested refactorings. (MC) is given by the following equation (7):

$$MC_{manual\_correctness} = \frac{|\textit{ relevant Web service refactorings}|}{|\textit{suggested Web service refactorings}|} \in [0:1] \quad (7)$$

In addition to the \textit{MC} measure, we have also evaluated the ability of our approach to fix design defects using the percentage of fixed defects NF which is the percentage of design defects fixed by the application of the best refactoring solution and corresponds as defined in (8); to the number of fixed defects divided by the total number of defects.

$$NF = \frac{\#\textit{ fixed design antipatterns}}{\#\textit{ design antipatterns}} \in [0:1] \quad (8)$$

The defects are detected using a set of rules defined in our previous work [4].

For the automatic validation we compared the proposed Web service refactorings with the expected ones using the traditional precision and recall evaluation measures [33]. These expected refactorings are suggested by users (e.g. subjects of our study) to fix existing Web service design defects as detailed later. The precision is to estimate the ratio of correctly Web services refactorings among the set of all proposed refactorings. We compute the precision score as follows (9):

$$RE_{precision} = \frac{|\textit{ suggested Web service refactorings} \cap \textit{expected Web service refactorings }|}{|\textit{ suggested refactorings}|} \in [0:1] \quad (9)$$

The recall denotes the ratio of correct refactorings selected among the set of all expected ones and is given by Eq. (10).

$$RE_{recall} = \frac{|\textit{ suggested Web service refactorings} \cap \textit{expected Web service refactorings }|}{|\textit{expected Web service refactorings}|} \in [0:1] \quad (10)$$

To investigate our second research question **RQ2**, we examine firstly the number of objectives (NOB) and then, we evaluate the efficiency of our approach to restructure Web services design in terms of precision, recall, manual correctness and percentage of fixed defects.

To answer **RQ3**, we compared our results with a recent state-of-the art approaches by [10], [11], [12]. Athanasopoulos et al. proposed a Web service refactoring approach based on a greedy algorithm to refactor and split Web service interfaces based on different cohesion measures. Ouni et al. proposed a graph decomposition approach for Web services remodularization using coupling and cohesion metrics. Wang et al. proposed a dimensionality reduction technique based on PCA-

NSGAII to address the Web services modularization problem using several Web service quality metrics.

To answer **RQ4**, we assess the quality metrics improvement of the interface remodularization solutions proposed by our approach for 570 Web services.

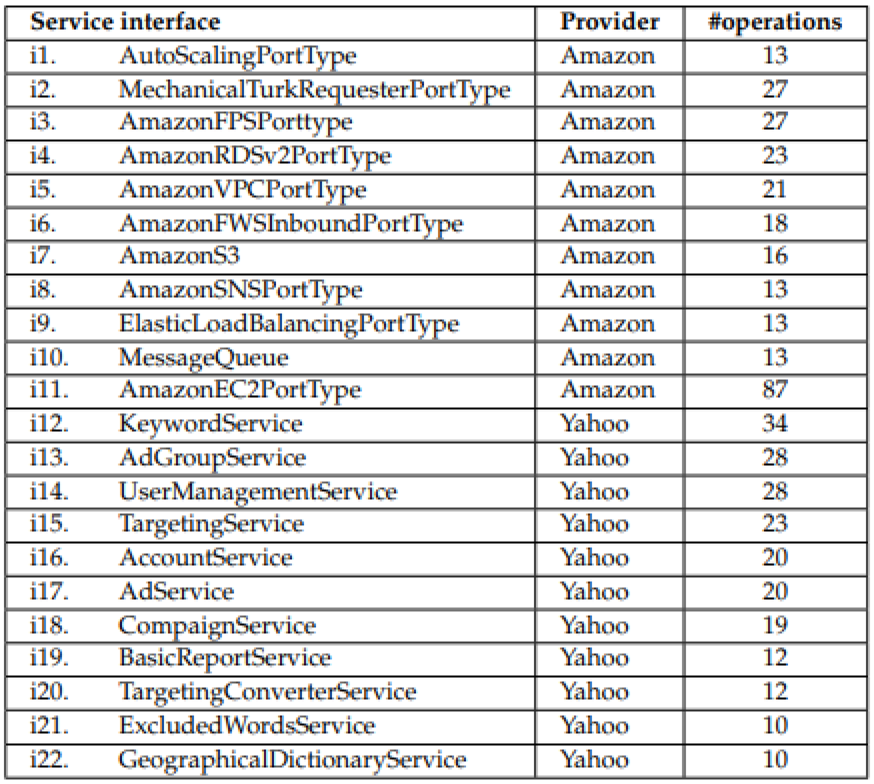
4.2 Experimental Setup

The aim of the experiment presented in this paper is to answer the previous research questions. Our evaluation involved 14 independent volunteer participants including 6 industrial developers and 8 graduate students. In particular, 3 senior developers from *Browser King*[5], 3 developers from *Accunet Web Services*[6], 3 MSc and 5 PhD candidates in Software Engineering. We first gathered information about the participant's background. All participants are familiar with service-oriented development and SOAP Web services with an experience ranging from 4 to 9 years. The participants were unaware of the techniques to be evaluated neither the particular research questions, in order to guarantee that there will be no bias in their judgment. In the following, we will describe Web services used.

We conducted our experiment on a benchmark of 22 real-world services provided by Amazon[7] and Yahoo[8]. We selected services with interfaces exposing at least 10 operations. We chose these Web services because their WSDL interfaces are publicly available, and they are widely used in different contexts and were previously studied in the literature [10], [34]. **Table 1** presents our used benchmark.

[5] http://www.browserkings.com
[6] www.accunet.us
[7] http://aws.amazon.com
[8] developer.searchmarketing.yahoo.com/docs/V6/reference/

TABLE 1: Amazon and Yahoo benchmark overview.



Service interface		Provider	#operations
i1.	AutoScalingPortType	Amazon	13
i2.	MechanicalTurkRequesterPortType	Amazon	27
i3.	AmazonFPSPorttype	Amazon	27
i4.	AmazonRDSv2PortType	Amazon	23
i5.	AmazonVPCPortType	Amazon	21
i6.	AmazonFWSInboundPortType	Amazon	18
i7.	AmazonS3	Amazon	16
i8.	AmazonSNSPortType	Amazon	13
i9.	ElasticLoadBalancingPortType	Amazon	13
i10.	MessageQueue	Amazon	13
i11.	AmazonEC2PortType	Amazon	87
i12.	KeywordService	Yahoo	34
i13.	AdGroupService	Yahoo	28
i14.	UserManagementService	Yahoo	28
i15.	TargetingService	Yahoo	23
i16.	AccountService	Yahoo	20
i17.	AdService	Yahoo	20
i18.	CompaignService	Yahoo	19
i19.	BasicReportService	Yahoo	12
i20.	TargetingConverterService	Yahoo	12
i21.	ExcludedWordsService	Yahoo	10
i22.	GeographicalDictionaryService	Yahoo	10

4.3 Results and Discussions

Results for RQ1. To answer RQ1, we evaluated the average of Manual Correctness *(MC)*, the percentage of fixed defects *(NF)*, Precision *(PR)* and Recall *(RC)* scores. **Figure 4** presents the evaluation of the *MC* score. We observe that the score is higher than 85% on all the 22 Web services. The correctness score achieved 100% on 10 Web services, which means that, from the developers' point of view the suggested Web services refactorings for these Web services are all correct. Thus, this finding indicates that the created port types and applied changes to the initial design are considered useful and correct according the developers of our experiments.

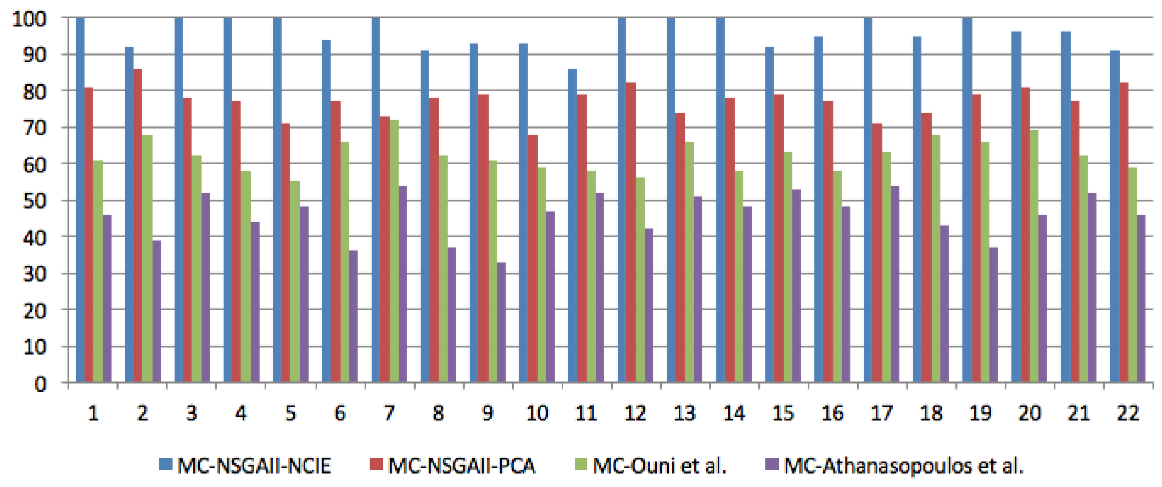


Figure 4: Median manual correctness (MC) value over 30 runs on all the Web services using the different techniques with a 95% confidence level ($\alpha < 5\%$).

We reported also the ability of our approach to fix several types of design defects and to improve the service interface design quality as described in **Figure 5** that depicts the results of the percentage of fixed several types of design defects *(NF)*. On average, for the different Web services, 85% of design defects are considered fixed by developers, which is considered a good score since developers may reject or modify some design changes that fix some defects because they do not consider some of them as very important (their goal is not to fix all design defects in the Web service interface) or because they wanted to focus on improving the cohesion and minimize coupling. We found that our approach succeeded to fix all design antipatterns of *AmazonVPCPortType* and *MessageQueue* services. The manual evaluation results by developers confirm that our approach produces relevant Web services refactoring suggestions. Nevertheless, these two measures just evaluate the correctness and not the relevance of the recommended solutions, therefore we also evaluated our approach automatically by comparing the proposed modularization changes with some expected ones defined manually by the different groups for the different Web services.

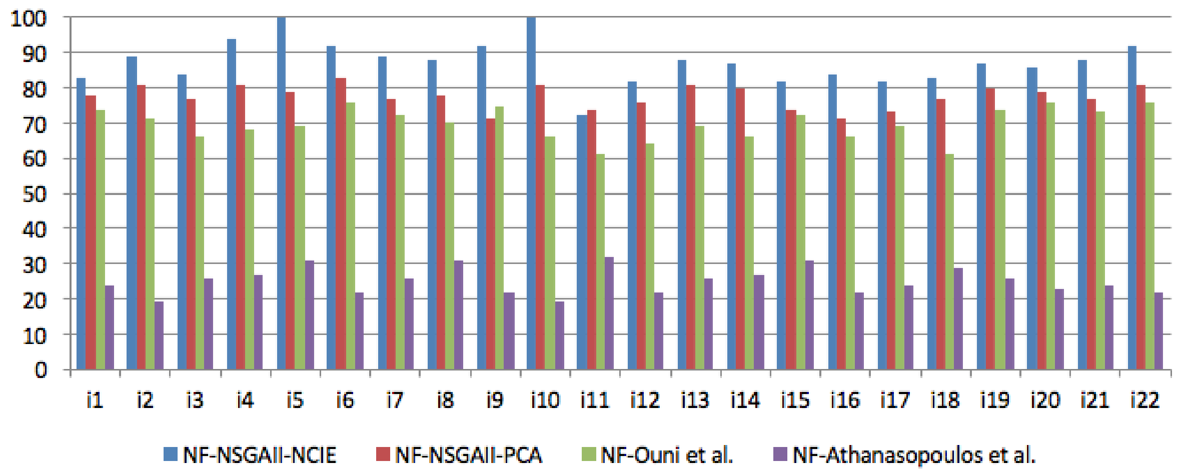


Figure 5: Median number of fixed design defects (NF) value over 30 runs on all the Web services using the different techniques with a 95% confidence level ($\alpha < 5\%$).

From **Figure 6**, we observe that NSGAII-NCIE performs well, with an average of precision of 81%, and a ratio of 97% of precision for *AutoScalingPortType* means that the results are very close to the actual defects, while is the lowest score was 58% for *AmazonEC2PortType*. This is not surprising since this Web service has also the worst MC, NF scores and it can be explained by the fact that this Web service include the higher number of operations than others (87 operations), thereby a higher number of antipatterns. In terms of recall, and as stated in **Figure 7**, we found that it differed only slightly between our Web services except few of them, the average recall value was 75%, i.e. is the proportion of design defects that were correctly identified, the highest observed value was 96% for *TargetingService* and the lowest was 65% for *AdGroupService* and *AdService*.

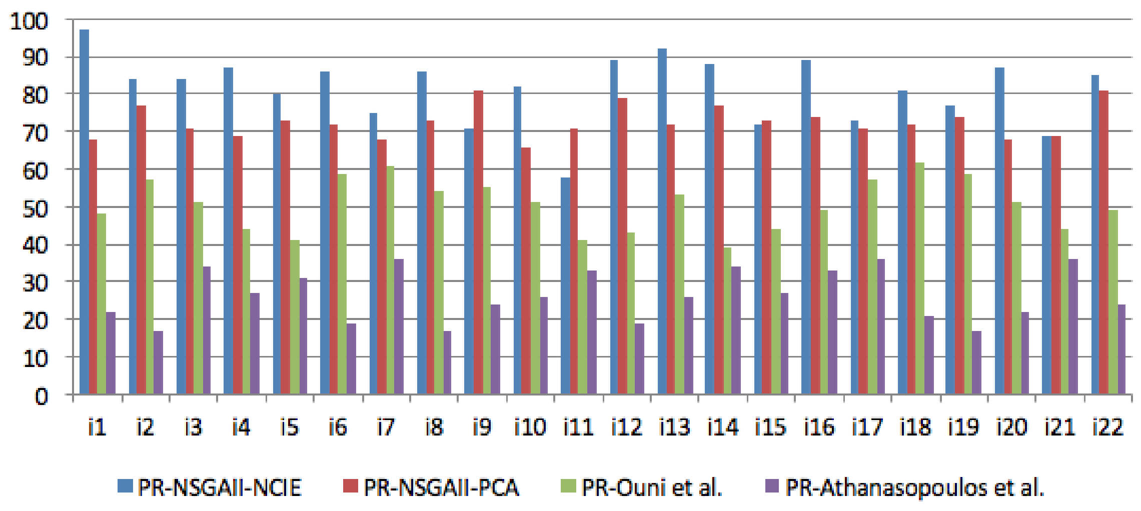


Figure 6: Median precision (PR) value over 30 runs on all the Web services using the different techniques with a 95% confidence level (α < 5%).

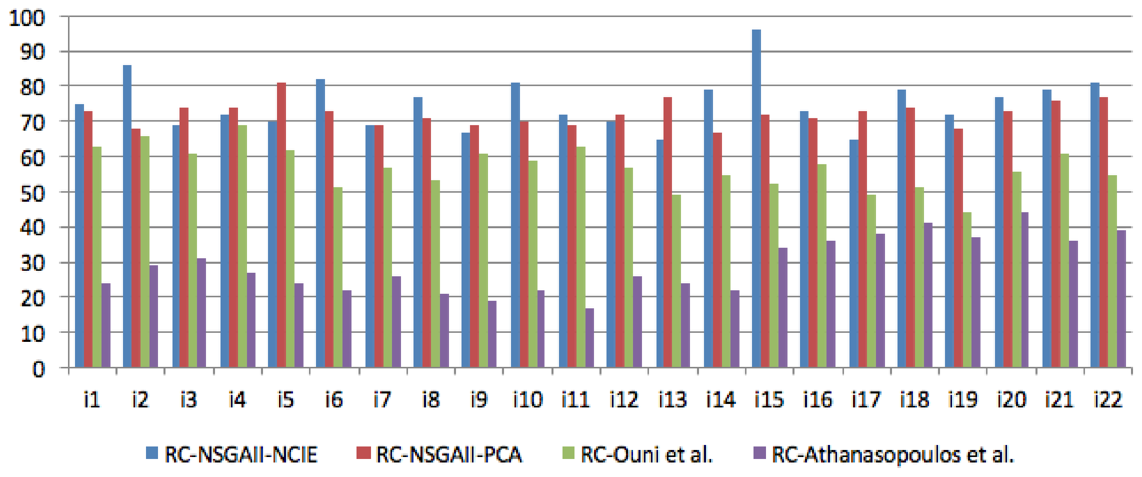


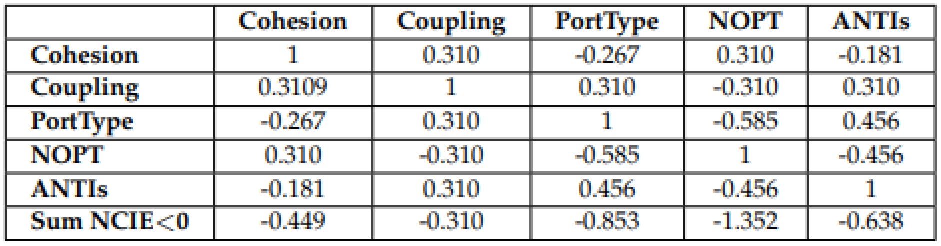
Figure 7: Median recall (RC) value over 30 runs on all the Web services using the different techniques with a 95% confidence level (α < 5%).

To summarize and answer RQ1, both automatic and manual experimentation results confirm that our approach helps the participants to restructure their Web service interface design efficiently by finding the relevant portTypes and improve the quality of all the 22 Web services.

Results for RQ2. These results need to be interpreted in the wake of: (i) the reduction of objectives, and (ii) the pertinence and the efficiency of our approach to restructure Web services design.

We present, here, how we selected the most conflicting objectives using the nonlinear correlation information entropy approach. **Table 2** gives an example of the process of our objective selection method. Firstly, the temporary set S_t is {*Cohesion, Coupling, PortType, NOPT, ANTIs*}, and the selected objective set S_r is Ø.

TABLE 2: Example of the modified NCIE matrix with 5 objectives.



	Cohesion	Coupling	PortType	NOPT	ANTIs
Cohesion	1	0.310	-0.267	0.310	-0.181
Coupling	0.3109	1	0.310	-0.310	0.310
PortType	-0.267	0.310	1	-0.585	0.456
NOPT	0.310	-0.310	-0.585	1	-0.456
ANTIs	-0.181	0.310	0.456	-0.456	1
Sum NCIE<0	-0.449	-0.310	-0.853	-1.352	-0.638

Objective *NOPT* is selected because it has the largest negative sum NCIE to other objectives (NOPT=-1.352) and *Cohesion* is omitted because it is positively correlated to the selected objective. In fact, *Cohesion* is a redundant objective with *NOPT*. Therefore, S_r = {*NOPT*} and S_t = {*Coupling, PortType, ANTIs*}. Then, objective *PortType* is selected because it is the most representative objective (PortType=0.767). *ANTIs* and *Coupling* are omitted because both objectives are positively correlated to the selected objective. Per consequent, the set of reduced and most conflicting objectives is {*PortType, NOPT*}.

Figure 8 shows the number of objectives after reduction over 30 independent runs on all systems.

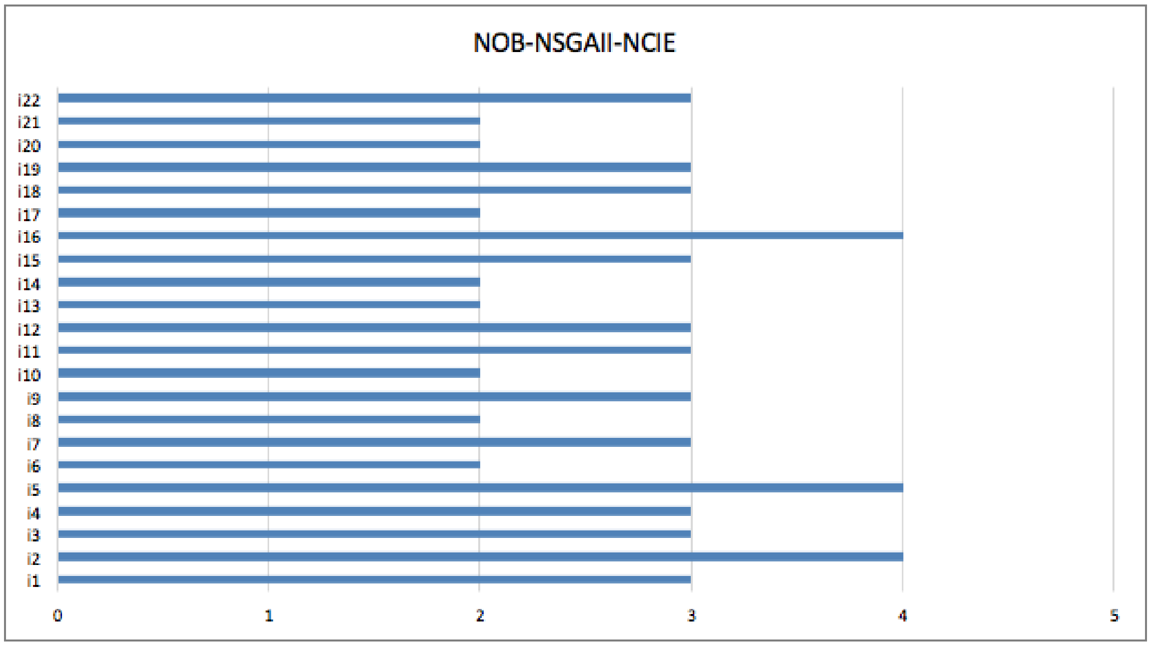


Figure 8: Median number of objectives (NOB) value over 30 runs on all the Web services using NSGAII-NCIE.

As we can see, the number of objectives were reduced to only 2 in 8 Web services, 3 in 11 others, while 4 objectives might be required to perfectly represent the results for the rest of Web services. Thus, this finding indicates the ability of NCIE approach to reduce significantly the number of objectives. However, the interpretation of the objective reduction alone is inadequate. Thus, combined with the results of **RQ1**, where our approach achieving significantly high precision and recall with a good correctness and a great percentage of fixed defects, accompanying this reduction can confirm the usefulness of our approach.

To summarize and answer **RQ2**, it is clear that the proposed NSGAII-NCIE formulation successfully reduced the number of objectives while generating useful Web services refactoring recommendations.

Results for RQ3. To answer **RQ3**, we evaluate the efficiency of our approach comparing to three other contributions of Athanasopoulos et al. [10], Ouni et al. [11] and Wang et al. [12]. The comparison is performed in terms of: (1) correctness of suggested refactorings, (2) the percentage of fixed defects that is calculated using defect detection rules, (3) precision, and (4) recall values. As described in **Figure 4**, for the 22 Web services, an interesting improvement was achieved in terms of correctness by using NSGAII-NCIE and provides an average of 96% of proposed remodularization operations is considered correct and meaningful. This score is significantly higher than the scores of the three other approaches having respectively only 45%, 62% and 77%, on average as *MC* scores on the different Web services. Therefore, our approach achieved much better results comparing to the three other approaches in terms of correctness.

Regarding the percentage of fixed defects, we noted that more than 87% of detected design defects were fixed as an average for all the Web services. By comparison, as depicted in **Figure 5**, the results for Athanasopoulos et al. is only 25%, for Ouni et al. is 70% and for Wang et al. is 77%. The underperformance of Athanasopoulos technique and slightly less Ouni technique in terms of percentage of fixed defects, can be explained by the fact that the main goal of these studies is not to mainly fix these defects. These studies are limited to the coupling and cohesion metrics which may not be sufficient to guide the modularization of Web services.

We compared the four approaches in terms of automatic validation $RE_{precision}$ and RE_{recall}, as depicted respectively in **figures 6** and **7**. For all the 30 runs of the 22 Web services, NSGAII-NCIE achieved an average precision of 81% and an average recall of 75%, while in NSGAII-PCA achieved 72% and 77% of precision and recall, respectively. These two approaches were much better than Ouni et al. approach which achieved 50% of precision and 55% of recall and Athansopoulos et al. approach achieves an average precision of 26% and an average recall of 39%.

To summarize, NSGAII-NCIE approach were slightly better than NSGAII-PCA in most of the cases, and significantly better than those obtained in [10], [11] in all studied Web services.

Results for RQ4. To answer RQ4, we present the obtained quality metrics improvement values for Cohesion, number per port types *(NPT)*, number of operations per port types *(NOPT)* and number of Antipatterns *(ANTIs)* that we calculated before and after remodularization and this for 570 Web services. From **Figure 9**, we noticed that cohesion metric has been exponentially increasing with an average of cohesion of the refactored interfaces of 0.892 while it was before 0.066. This increase is due to the move of operations between service interfaces to have more cohesive services. Another interesting observation was that NPT metric has been growing, it was $\simeq 1$ as an average before and achieved $\simeq 8$ port types after the remodularization. Increasing the number of port types is a desired effect to avoid having all operations in a single large interface. Moreover, we have noted that the applied remodularization tend to produce smaller interfaces, it generated few operations per interfaces. The average value was 27 operations per port types before and only 5 after NSGAII-NCIE.

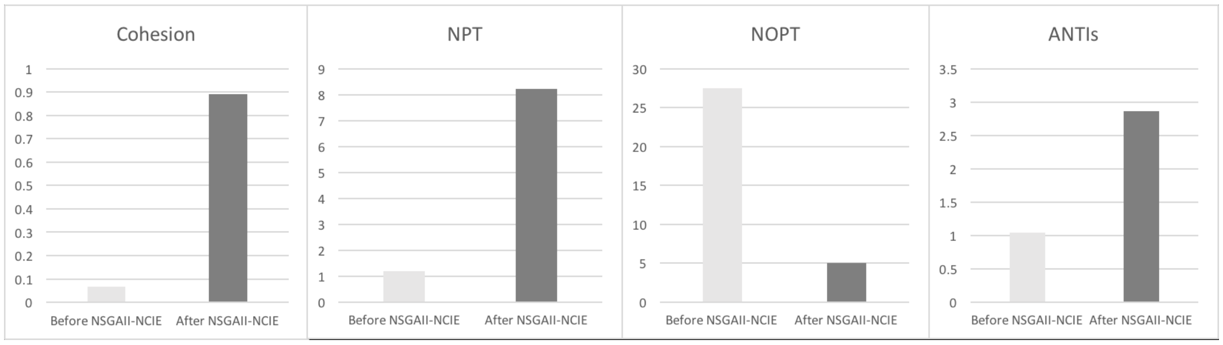


Figure 9: Quality Metrics Improvements using NSGAII-NCIE.

CHAPTER 5: THREATS TO VALIDITY

This study suffers from some threats to validity which prevent to generalize the obtained results such as the data collection. These threats to validity can be classified in three categories: construct, internal and external validity.

Construct threats to validity regard the relationship between theory and observation. A construct threat can be related to the number of selected objectives. Although we selected five quality metrics in this study, using more sets of metrics may prove fruitful. Another possible threat related to the suitability to our evaluation metrics. We used the standard metrics precision and recall that are widely accepted as good measures for antipattern detection solutions [4], [12]. Another construct threats to validity is the convenience of the threshold we applied to determine whether two objectives are correlated. In fact, the number of objectives per clusters selected by the widely known clustering algorithm K-means used could affect the results of our study e.g., some conflicting objectives would not be selected. But, classification for correlated and non-correlated objectives is a one-dimensional problem and K-means is an efficient algorithm to fulfill this task.

Internal threats to validity concern possible bias with the results obtained by our proposal. A threat to consider can be related to the knowledge and expertise of the human evaluators. Inadequate knowledge could lead to limited ability to assess the quality of an interface. We mitigate this threat by selecting participants having from 4 to 9 years of experience with service-oriented development and familiar with SOAP Web services. Moreover, to avoid bias in the experiment none of the authors have been involved in this evaluation. In addition, we randomized the ordering in which the MOWSIR, Athanasopoulos et al. and random refactorings were shown to participants, to mitigate any sort of learning or fatigue effect.

External threats to validity refers to the generalizability of our findings. In this study, it concerns mainly the studied Web services. We have selected 22 real world Web services belong to different application domains, offer diverse functionalities, have different sizes ranging from 10 to 87 operations and were developed by different companies Yahoo and Amazon. However, we considered in our approach only SOAP Web services and we cannot assert that our results can be generalized to services developed with other technologies e.g., REST Web services, and to other practitioners. In addition, we did not evaluate our approach on all possible defect types. However, the five types of Web service antipatterns we employed constitute a wide representative set of standard and most frequent defects.

CHAPTER 6: RELATED WORK

6.1 Web Services Design Quality

Detecting, fixing and specifying design defects in SOA and Web services is a relatively new area. The first book in the literature was written by Dudney et al. [19] and provides informal definitions of a set of Web service design defects. More recently, Rotem-Gal-Oz et al. [17] have provided the symptoms of some additional SOA antipatterns also informally. Furthermore, Král et al. [18] have described seven "popular" SOA antipatterns which violate the SOA principles but they did not discuss their detection. In another study, Rodriguez et al. [35] have introduced a catalog of WSDL-based Web Services discoverability antipatterns. Likewise, Torkmani et al. [36] have presented a repository of 45 general antipatterns in SOA. They have introduced a new method based on check lists to identify and detect antipatterns of SOA in service-oriented development. The goal of this work is a comprehensive review of these antipatterns that will help developers to work with a clear understanding of patterns in phases of software development and so avoid many potential problems. Moha et al. [37] have proposed a rule-based approach called SODA for SCA systems (Service Component Architecture). Later, Palma et al. [38] extended this work for Web service design defects in SODA-W. The proposed approach relies on declarative rule specification using a domain-specific language (DSL) to specify/identify the key symptoms that characterize design defect using a set of WSDL metrics. In another study, Rodriguez et al. [39], [40] and Mateos et al. [41] provided a set of guidelines for service providers to avoid bad practices while writing WSDLs. Based on some heuristics, the authors detected eight bad practices in the writing of WSDL for Web services. The heuristics are simple rules based on pattern matching. In other work [36], the authors presented a repository of 45 general design defects in SOA. The goal of this work is a comprehensive review of these defects that will help developers to work with a clear understanding of patterns in phases of software development and so avoid many potential problems. Recently, Ouni et al. [42], [43] proposed a search-based approach based on standard GP to find regularities, from examples of Web service design defects, to be translated into detection rules.

6.2 Services Interface Remodularization

Few studies have been focused on the problem of fixing Web service design antipatterns in the recent years. Remodularization of web service interface was first proposed by Athanasopoulos et al. [10]. They have proposed a single-objective optimization-based approach using greedy algorithm to automatically decomposing Web services interface into more cohesive interfaces in order to improve the service interface design quality. They considered only a suite of cohesion metrics which used interface specification. However, the notion of coupling has not been considered which is another main attribute related to the decomposition. Later, Ouni et al. suggested that coupling is as important metric as cohesion and proposed Service Interface Modularization (*SIM*) approach with an objective function formulated based on maximizing the interface cohesion and minimizing inter-interface coupling to automatically decomposing large interfaces with semantically unrelated operations into smaller cohesive interfaces [11]. In a recent study of Wang et al., an interactive recommendation approach was introduced [13]. This approach is based on interactive non-dominated sorting genetic algorithm (NSGA-II) to find the best trade-offs between these three objectives: (1) improving several interface design quality metrics (e.g. coupling, cohesion, number of portTypes and number of antipatterns) (2) maximizing the satisfaction of the interaction constraints learnt from the user/developer feedback, while (3) minimizing the deviation from the initial design. However, the proposed approach didn't support a high number of objectives.

Conversely, in the object paradigm, many works were conducted to improve the modular structure of the software and various studies have been published on that topic [44], [45], [46], [47], [48], [49], [50], [51], [52], [53]. Most of the proposed approaches to object-oriented systems remodularization are based on clustering algorithms or search-based techniques. We present some of the many existing approaches, focusing on applications of clustering or SBSE approaches to the problem of re-modularization.

Wiggerts provides an overview of software clustering techniques in systems Remodularization [44]. He discusses on how approach based on clustering algorithms automatically partition systems into cohesive clusters. Harman et al. [45] introduces a new representation and crossover operator for a software clustering problem, which reduces the search space of possible modularizations. Seng et al. treated the remodularization task as a single-objective optimization problem using genetic algorithm to improve the decomposition of a software system [46].

Since 1998, SBSE has been applied extensively to support the automation of software modularization. The first attempt to address this problem using search-based approach was presented by Mancoridis et al. [48]. In a later study, they experimented their Bunch tool [49] for module clustering with genetic algorithms, hill climbing and simulated annealing. A more recent work by Bavota et al. [51] introduces an automatic approach to determine the related classes that should belong together in a package by combining use of semantic and structural measures. Praditwong et al. formulated the clustering problem as a multi-objective optimization problem [50], in which the goal is to maximizing the sum of intra-edges of all clusters, minimizing the sum of inter-edges of all clusters, maximizing the number of clusters, maximizing the number of isolated clusters and minimizing the number of isolated clusters. Later, Abdeen et al. [52] used multi-objective optimization approach based on NSGA-II in order to improve packages structure. Their approaches use the following objectives: (1) maximizing package cohesion, (2) minimizing package coupling, (3) minimizing package cycles, (4) avoiding Blob packages, and (5) minimizing the modification of the original modularization. More recently, in 2015, Mkaouer et al. proposed a novel many-objective search-based approach using NSGA-III for the automation of software remodularization [53] as the number of objectives considered in their problem formulation is high such as structural improvement of packages, number of changes, semantic coherence of the restructured program, and consistency with change history.

CHAPTER 7: CONCLUSION

In this paper, we proposed a dimensionality reduction approach for multi-objective Web services remodularization that adjusts the number of considered objectives during the search for near optimal solutions. A regular multi-objective NSGA-II algorithm with an initial set of five quality metrics is executed for a number of iterations then we employ the nonlinear metric NCIE to measure the correlation among objectives. The number of objectives maybe reduced during the next iterations based on NCIE results. The process is repeated several times until a maximum number of iterations is reached to generate a set of non-dominated Web services modularization solutions. The proposed approach is evaluated on (i) 22 real-world Web services provided by Amazon and Yahoo; where we conducted a human study on a set of users who evaluated the approach and compared it with the state-of-the-art Web services modularization techniques and (ii) a benchmark of 570 Web services where we evaluated the improvements of quality metrics. Our evaluation results provide strong evidence that our technique successfully reduced significantly the initial set of large number of objectives/quality metrics. The results also show that our approach outperforms several of existing Web services modularization techniques [10], [11], [12]. In our future work, we are planning to consider a larger set of refactorings in our experiments and to validate our technique with additional objectives, antipatterns and Web services in order to conclude about the general applicability of our methodology. Furthermore, we are planning to adapt our dimensionality reduction approach to others problems such as Web services composition. Another future research direction related to our work is to integrate the developers in the loop when reducing the number of objectives to either select which one to eliminate or revise the fitness function formulation (aggregating the objectives).

REFERENCES

[1] Leymann, F., Roller, D., & Schmidt, M. T. (2002). Web services and business process management. *IBM systems Journal*, *41*(2), 198-211.

[2] Romano, D., & Pinzger, M. (2012, June). Analyzing the evolution of web services using fine-grained changes. In *Web Services (ICWS), 2012 IEEE 19th International Conference on* (pp. 392-399). IEEE.

[3] Papazoglou, M. P., & Van Den Heuvel, W. J. (2006). Service-oriented design and development methodology. *International Journal of Web Engineering and Technology*, *2*(4), 412-442.

[4] Ouni, A., Kessentini, M., Inoue, K., & Cinnéide, M. O. (2017). Search-based web service antipatterns detection. *IEEE Transactions on Services Computing*, *10*(4), 603-617.

[5] Perepletchikov, M., Ryan, C., & Tari, Z. (2010). The impact of service cohesion on the analyzability of service-oriented software. *IEEE Transactions on Services Computing*, *3*(2), 89-103.

[6] Wang, H., Kessentini, M., & Ouni, A. (2016, October). Bi-level identification of web service defects. In *International Conference on Service-Oriented Computing* (pp. 352-368). Springer, Cham.

[7] Ouni, A., Daagi, M., Kessentini, M., Bouktif, S., & Gammoudi, M. M. (2017, June). A Machine Learning-Based Approach to Detect Web Service Design Defects. In *Web Services (ICWS), 2017 IEEE International Conference on* (pp. 532-539). IEEE.

[8] Moha, N., Palma, F., Nayrolles, M., Conseil, B. J., Guéhéneuc, Y. G., Baudry, B., & Jézéquel, J. M. (2012, November). Specification and detection of SOA antipatterns. In *International Conference on Service-Oriented Computing* (pp. 1-16). Springer, Berlin, Heidelberg.

[9] Palma, F., Nayrolles, M., Moha, N., Guéhéneuc, Y. G., Baudry, B., & Jézéquel, J. M. (2013). Soa antipatterns: An approach for their specification and detection. *International Journal of Cooperative Information Systems*, *22*(04), 1341004.

[10] Athanasopoulos, D., Zarras, A. V., Miskos, G., Issarny, V., & Vassiliadis, P. (2015). Cohesion-driven decomposition of service interfaces without access to source code. *IEEE Transactions on Services Computing*, *8*(4), 550-562.

[11] Ouni, A., Salem, Z., Inoue, K., & Soui, M. (2016, June). SIM: an automated approach to improve web service interface modularization. In *Web Services (ICWS), 2016 IEEE International Conference on* (pp. 91-98). IEEE.

[12] Wang, H., & Kessentini, M. (2017, November). Improving Web Services Design Quality Using Dimensionality Reduction Techniques. In *International Conference on Service-Oriented Computing* (pp. 499-507). Springer, Cham.

[13] Wang, H., Kessentini, M., & Ouni, A. (2017). Interactive Refactoring of Web Service Interfaces Using Computational Search. *IEEE Transactions on Services Computing*.

[14] D. Brockhoff and E. Zitzler, "Are all objectives necessary? on dimensionality reduction in evolutionary multiobjective optimization," in Parallel Problem Solving from Nature-PPSN IX. Springer, 2006, pp. 533–542.

[15] Saxena, D. K., Duro, J. A., Tiwari, A., Deb, K., & Zhang, Q. (2013). Objective reduction in many-objective optimization: Linear and nonlinear algorithms. *IEEE Transactions on Evolutionary Computation*, *17*(1), 77-99.

[16] Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. A. M. T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE transactions on evolutionary computation*, *6*(2), 182-197.

[17] A. Rotem-Gal-Oz, SOA Patterns. Manning Publications, 2012

[18] Kral, J., Zemlicka, M., & Popular, S. O. A. Antipatterns, Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns, 2009. *COMPUTATIONWORLD*, *9*, 271-276.

[19] B. Dudney, J. Krozak, K. Wittkopf, S. Asbury, and D. Osborne, J2EE Antipatterns. John Wiley; Sons, Inc., 2003.

[20] A. Rotem-Gal-Oz, E. Bruno, and U. Dahan, SOA patterns. Manning, 2012.

[21] Palma, F., Moha, N., Tremblay, G., & Guéhéneuc, Y. G. (2014, August). Specification and detection of SOA antipatterns in web services. In *European Conference on Software Architecture* (pp. 58-73). Springer, Cham.

[22] M. Perepletchikov, C. Ryan, K. Frampton, and Z. Tari, "Coupling metrics for predicting maintainability in service-oriented designs," in Software Engineering Conference, 2007. ASWEC 2007. 18th Australian. IEEE, 2007, pp. 329–340.

[23] K. Deb and D. K. Saxena, "On finding pareto-optimal solutions through dimensionality reduction for certain large dimensional multi-objective optimization problems," Kangal report, vol. 2005011, 2005.

[24] K. Deb and H. Jain, “Handling many-objective problems using an improved nsga-ii procedure,” in Evolutionary computation (CEC), 2012 IEEE congress on. IEEE, 2012, pp. 1–8.

[25] A. L. Jaimes, C. A. C. Coello, and J. E. U. Barrientos, “Online objective reduction to deal with many-objective problems,” in International Conference on Evolutionary Multi-Criterion Optimization. Springer, 2009, pp. 423–437.

[26] S. Kukkonen and J. Lampinen, “Ranking-dominance and manyobjective optimization,” in Evolutionary Computation, 2007. CEC 2007. IEEE Congress on. IEEE, 2007, pp. 3983–3990.

[27] López Jaimes, A., Coello Coello, C. A., & Chakraborty, D. (2008, July). Objective reduction using a feature selection technique. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation* (pp. 673-680). ACM.

[28] D. Brockhoff and E. Zitzler, “Objective reduction in evolutionary multiobjective optimization: Theory and applications,” Evolutionary computation, vol. 17, no. 2, pp. 135–166, 2009.

[29] A. R. Freitas, P. J. Fleming, and F. G. Guimaraes, “A nonparametric harmony-based objective reduction method for manyobjective optimization,” in Systems, Man, and Cybernetics (SMC), 2013 IEEE International Conference on. IEEE, 2013, pp. 651–656.

[30] H. Wang and X. Yao, “Objective reduction based on nonlinear correlation information entropy,” Soft Computing, vol. 20, no. 6, pp. 2393–2407, 2016.

[31] K. Mitra and R. Gopinath, “Multiobjective optimization of an industrial grinding operation using elitist nondominated sorting genetic algorithm,” Chemical Engineering Science, vol. 59, no. 2, pp. 385–396, 2004.

[32] Q. Wang, Y. Shen, and J. Q. Zhang, “A nonlinear correlation measure for multivariable data set,” Physica D: Nonlinear Phenomena, vol. 200, no. 3-4, pp. 287–295, 2005.

[33] D. L. Olson and D. Delen, Advanced data mining techniques. Springer Science & Business Media, 2008.

[34] M. Fokaefs, R. Mikhaiel, N. Tsantalis, E. Stroulia, and A. Lau, “An empirical study on web service evolution,” in Web Services (ICWS), 2011 IEEE International Conference on. IEEE, 2011, pp. 49–56.

[35] J. M. Rodriguez, M. Crasso, A. Zunino, and M. Campo, “Improving web service descriptions for effective service discovery,” Science of Computer Programming, vol. 75, no. 11, pp. 1001–1021, 2010.

[36] M. A. Torkamani and H. Bagheri, “A Systematic Method for Identification of Anti-patterns in Service Oriented System Development,” International Journal of Electrical and Computer Engineering, vol. 4, no. 1, pp. 16–23, 2014.

[37] N. Moha, F. Palma, M. Nayrolles, B. J. Conseil, Y.-G. Gueh´ eneuc, ´ B. Baudry, and J.-M. Jez´ equel, ´ Service-Oriented Computing: 10th International Conference, ICSOC 2012, Shanghai, China, November 12- 15, 2012. Proceedings. Springer Berlin Heidelberg, 2012, pp. 1–16.

[38] F. Palma, N. Moha, G. Tremblay, and Y.-G. Gueh´ eneuc, “Specifica- ´ tion and detection of soa antipatterns in web services,” in Software Architecture. Springer, 2014, pp. 58–73.

[39] J. M. Rodriguez, M. Crasso, A. Zunino, and M. Campo, “Automatically detecting opportunities for web service descriptions improvement,” in Software Services for e-World. Springer, 2010, pp. 139–150.

[40] A. Sinha, P. Malo, A. Frantsev, and K. Deb, “Multi-objective stackelberg game between a regulating authority and a mining company: A case study in environmental economics,” in IEEE Congress on Evolutionary Computation, 2013, pp. 478–485.

[41] C. Mateos, A. Zunino, and J. L. O. Coscia, “Avoiding WSDL Bad Practices in Code-First Web Services,” SADIO Electronic Journal of Informatics and Operational Research, vol. 11, no. 1, pp. 31–48, 2012.

[42] A. Ouni, M. Kessentini, K. Inoue, and M. O Cinneide, “Searchbased web service antipatterns detection,” IEEE Transactions on Services Computing, vol. PP, no. 99, 2015.

[43] A. Ouni, R. Gaikovina Kula, M. Kessentini, and K. Inoue, “Web service antipatterns detection using genetic programming,” in Conference on Genetic and Evolutionary Computation (GECCO), ser. GECCO ’15, 2015, pp. 1351–1358.

[44] T. A. Wiggerts, “Using clustering algorithms in legacy systems remodularization,” in Reverse Engineering, 1997. Proceedings of the Fourth Working Conference on. IEEE, 1997, pp. 33–43.

[45] M. Harman, R. Hierons, and M. Proctor, “A new representation and crossover operator for search-based optimization of software modularization,” in Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation. Morgan Kaufmann Publishers Inc., 2002, pp. 1351–1358.

[46] O. Seng, M. Bauer, M. Biehl, and G. Pache, “Search-based improvement of subsystem decompositions,” in Proceedings of the 7th annual conference on Genetic and evolutionary computation. ACM, 2005, pp. 1045–1051.

[47] O. Maqbool and H. Babri, “Hierarchical clustering for software architecture recovery,” IEEE Transactions on Software Engineering, vol. 33, no. 11, 2007.

[48] S. Mancoridis, B. S. Mitchell, C. Rorres, Y. Chen, and E. R. Gansner, "Using automatic clustering to produce high-level system organi- 13 zations of source code," in Program Comprehension, 1998. IWPC'98. Proceedings., 6th International Workshop on. IEEE, 1998, pp. 45–52.

[49] B. S. Mitchell and S. Mancoridis, "On the automatic modularization of software systems using the bunch tool," IEEE Transactions on Software Engineering, vol. 32, no. 3, pp. 193–208, 2006.

[50] K. Praditwong, M. Harman, and X. Yao, "Software module clustering as a multi-objective search problem," IEEE Transactions on Software Engineering, vol. 37, no. 2, pp. 264–282, 2011.

[51] G. Bavota, A. De Lucia, A. Marcus, and R. Oliveto, "Software re-modularization based on structural and semantic metrics," in Reverse Engineering (WCRE), 2010 17th Working Conference on. IEEE, 2010, pp. 195–204.

[52] H. Abdeen, H. Sahraoui, O. Shata, N. Anquetil, and S. Ducasse, "Towards automatically improving package structure while respecting original design decisions," in Reverse Engineering (WCRE), 2013 20th Working Conference on. IEEE, 2013, pp. 212–221.

[53] W. Mkaouer, M. Kessentini, A. Shaout, P. Koligheu, S. Bechikh, K. Deb, and A. Ouni, "Many-objective software remodularization using nsga-iii," ACM Transactions on Software Engineering and Methodology (TOSEM), vol. 24, no. 3, p. 17, 2015.