The current issue and full text archive of this journal is available on Emerald Insight at:
https://www.emerald.com/insight/1744-0084.htm

A URI parsing technique and algorithm for anti-pattern detection in RESTful Web services

Fuad Sameh Alshraiedeh
Jordan Ministry of Education, Amman, Jordan and School of Computing, Universiti Utara Malaysia, Sintok, Malaysia, and
Norliza Katuk
School of Computing, Universiti Utara Malaysia, Sintok, Malaysia

Received 18 August 2020
Revised 18 October 2020
Accepted 20 October 2020

Abstract

Purpose – Many REpresentational State Transfer (RESTful) Web services suffered from anti-patterns problem, which may diminish the sustainability of the services. The anti-patterns problem could happen in the code of the programme or the uniform resource identifiers (URIs) of RESTful Web services. This study aims to address the problem by proposing a technique and an algorithm for detecting anti-patterns in RESTful Web services. Specifically, the technique is designed based on URIs parsing process.

Design/methodology/approach – The study was conducted following the design science research process, which has six activities, namely, identifying problems, identifying solutions, design the solutions, demonstrate the solution, evaluation and communicate the solution. The proposed technique was embedded in an algorithm and evaluated in four phases covering the process of extracting the URIs, implementing the anti-pattern detection algorithm, detecting the anti-patterns and validating the results.

Findings – The results of the study suggested an acceptable level of accuracy for the anti-patterns detection with 82.30% of precision, 87.86% of recall and 84.93% of F-measure.

Practical implications – The technique and the algorithm can be used by developers of RESTful Web services to detect possible anti-pattern occurrences in the service-based systems.

Originality/value – The technique is personalised to detect amorphous URI and ambiguous name anti-patterns in which it scans the Web service URIs using specified rules and compares them with pre-determined syntax and corpus.

Keywords Information services, Service-oriented architecture, Web services, Distributed computer systems, Resource-oriented architecture, Software architecture, Quality of service, Anti-patterns, Semantic Web, Schema, Semantic

Paper type Research paper

1. Introduction

Service-based system (SBS) is a type of software system that offers certain functions using Web services (WSs). They are increasingly popular as a result of the widespread use of service-oriented computing and cloud computing (Xie *et al.*, 2019). SBS is an essential part of various business functions, where elements within a system can interact with various other external systems (Aradea *et al.*, 2020). The ability to reuse services has been the primary reason for the

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

1744-0084
DOI 10.1108/IJWIS-08-2020-0052

adoption of service-oriented architecture (SOA) (Hamzah *et al.*, 2019; Hamzah *et al.*, 2018; Ahmad *et al.*, 2017). SOA has an open, flexible and independent architecture that functions autonomously and is able to be reused in the form of WSs (Abidi *et al.*, 2019). The increase in WS popularity and adaption is because of their flexibility in offering software services over the internet (Alshraiedeh and Katuk, 2016; Ma *et al.*, 2017). WS appears in two architectural styles: Simple Object Access Protocol (SOAP) and REpresentational State Transfer (RESTful). In the past few years, RESTful WS stands out as a mainstream technology (Ma *et al.*, 2018, 2017) because of its features such as resource representation hiding, run-time dynamic service binding, languages interoperability and flexible invocation (Brabra *et al.*, 2019; Fokaefs and Stroulia, 2015). RESTful WS can be either dynamic or static, where dynamic WS resources can be shared or isolated from the Web environment at different time. On the other hand, static WS resources are always available in the Web environment for open accessibility (Kallab *et al.*, 2019; Palma, 2015).

SBS has changed the way software systems are developed, published and consumed (Alshraiedeh *et al.*, 2015; Palma *et al.*, 2015). For example, companies such as Google, Facebook, Netflix, eBay, LinkedIn, Foursquare and Instagram have made their WS public to external users for their use (Ma *et al.*, 2018). RESTful architecture style exposes services as resource invocations which are identified by uniform resource identifiers (URIs) (Ruokonen *et al.*, 2016). It uses a uniform interface programmed in Hypertext Transfer Protocol (HTTP) to access and modify the resources (Huang and Wu, 2016; Petrillo *et al.*, 2018). The URI forms a RESTful indexing schema that matches resources to their provided functions (Kallab *et al.*, 2019) which make the resources are addressable and have a simple map format (Ma *et al.*, 2018; Aghajani *et al.*, 2018; Palma, 2015; Palma *et al.*, 2017).

The growing popularity of WS requires efficient mechanisms and tools to facilitate the matching, publishing, discovery and translation of their functions (Kehagias *et al.*, 2018). A RESTful WS is accessible through its interface that mapped to the corresponding resources (Rodriguez *et al.*, 2015; Xue *et al.*, 2015). However, many WSs suffer from design issues such as anti-patterns. One of the typical poor RESTful WS design practices is using unrelated names for resources or parameters and then group them in service (Aghajani *et al.*, 2018; Palma *et al.*, 2017). Consequently, such services are difficult to understand and reuse by other users (or programmes) and often result in underused services (Boukharata *et al.*, 2019; Ouni *et al.*, 2016). Indeed, to guarantee the successful utilisation of the services, users (or programmes) must understand the functions of the services before they use them (Ouni *et al.*, 2016). On the other hand, utilisation of poorly designed services will result in an unexpected response or outcome (Ouni *et al.*, 2019). Therefore, it is essential to use clear and relevant names of resources, services and parameters in the design of RESTful WSs (Palma *et al.*, 2015). The use of ambiguous naming is an example of anti-patterns, which may degrade the overall design and functionality of RESTful WSs (Palma *et al.*, 2015). The situation will affect the usage of the RESTful WSs, which consequently diminishes the sustainability of the services.

This study intends to bridge the gap of anti-patterns problem found in RESTful WSs. Anti-patterns can be detected either manually or automatically by experienced software or service developers. However, the literature analysis on RESTful WS anti-pattern detection suggested that the topic is still in its infancy stage (Nayrolles *et al.*, 2013; Rai *et al.*, 2015). Many aspects of the anti-patterns have not been adequately addressed, including proper and accurate detection of the anti-patterns. Based on the current development in WS anti-patterns, there is a need to study practical and accurate detection techniques. Accordingly, this study proposed a technique and an algorithm for detecting anti-patterns in URI RESTful WSs. The results of the study aim to facilitate service developers in detecting anti-patterns at the early stage of RESTful WS development.

2. Related work

WS design quality is an essential aspect of developing understandable and usable SBSs (Brabra *et al.*, 2019). Software quality assurance would be the expected result of software developers practice during the development process (De Renzis *et al.*, 2017; Mateos *et al.*, 2015a). Brabra *et al.* (2019) studied the widely used RESTful WSs in terms of URIs space design, resources representation and hyper-links support. On the other hand, Ruokonen *et al.* (2016) evaluated the compliance of best and poor design practices in RESTful WS from the perspective of mobile applications. Palma *et al.* (2015) evaluated the linguistic aspects of RESTful WS documentation in terms of best and poor practices of design in social and cloud platforms such as Facebook, Twitter, Dropbox and Bestbuy. Petrillo *et al.* (2018) evaluated three types of cloud RESTful WS using a catalogue of 73 general best practices. Service anti-patterns have been the standard means to evaluate software system design quality (Brabra *et al.*, 2019) which will lead to patterns and anti-pattern.

Patterns are defined as good practices, whether generic or specialised in software development and management, and they are the solutions to the common software issues (Stamelos, 2010). On the other hand; Stamelos (2010) defined anti-patterns as problematic software conditions as a result of human error or social and cultural related issues that may expose software to various risks. Just like any other computer-based system, anti-patterns are also found in WSs because of improper service design and implementation as well as programming practices (Alshraiedeh and Katuk, 2019; Kumar and Sureka, 2018). Many anti-patterns have been discovered in RESTful WS; for example, ambiguous names, amorphous URIs, CRUD URIs, contextless resource names and non-hierarchical nodes. These anti-patterns have their counterpart patterns. For example, amorphous URI is an anti-pattern, whereas tidy URI is the pattern. Figure 1 shows the commonly found patterns and their anti-patterns in RESTful WS as defined by Palma *et al.* (2015).

In the past few years, researchers conducted studies to address anti-pattern issues found in RESTful WSs. For example, Petrillo *et al.* (2018) designed the CloudLex approach for extracting and analysing four anti-patterns in RESTful WSs for cloud computing lexicons. Palma *et al.* (2015) surveyed the literature and presented a catalogue of 73 best practices in designing high-quality RESTful application programming interfaces for making them more understandable and reusable. Other works such as Lizarralde *et al.* (2019) and Mateos *et al.* (2015b) defined new anti-patterns contributing to the service anti-patterns catalogue. As the catalogue grows, there is a need for approaches to specify and detect anti-patterns automatically regardless of their underlying platform.

However, studies on WS anti-pattern detection are still in its infancy stage (Nayrolles *et al.*, 2013; Rai *et al.*, 2015) where many aspects of the anti-patterns have not adequately addressed. Nonetheless, researchers have started proposing algorithms and methods for the automated detection of anti-patterns. Palma *et al.* (2014a) proposed an approach named SODA-R to detect five patterns and eight anti-patterns in RESTful WS using heuristics approach. In another study by Palma *et al.* (2014b), SODA-W was proposed to detect ten types of well-known anti-patterns. Palma *et al.* (2015) and Palma *et al.* (2017) proposed DOLAR and SARA, respectively, to detect linguistics anti-patterns in RESTful WS. Studies have been performed to detect the anti-patterns from software performance perspective such as Sabir *et al.* (2017), Palma *et al.* (2017), Kalyani (2018), Saluja and Batra (2019). Relevant studies on WS anti-patterns detection are listed in Table 1.

☑ Patterns → ☒Anti-patterns
☑Contextualised Resource Names → ☒Contextless Resource Names
☑Hierarchical Nodes → ☒Non-hierarchical Nodes
☑Tidy URIs →☒Amorphous URIs
☑Verbless URIs →☒CRUDy URIs
☑Singlurised Nodes →☒Pluralised Nodes

Figure 1.
RESTful WS patterns and anti-patterns

Table 1. Studies of anti-patterns detection

Studies	Approach/tool	Setting	Architecture style	Types of anti-pattern
Atlidakis *et al.* (2019)	RESTler	Azure and Office365 cloud services	REST APIs	Bugs (i.e. 500 HTTP status code)
Brabra *et al.* (2019)	Semantic-based approach	Cloud REST APIs	Cloud APIs	Modularity-related anti-patterns such as multi-service, nano-service, chatty and data service interfaces
Saluja and Batra (2019)	Detection approach of anti-pattern based on static metrics	Web service description files	SOAP	Binary large object (BLOB) and Swiss Army Knife
Kalyani (2018)	Detection framework	Two benchmark data sets	SOA	Search-based anti-patterns
Hirsch *et al.* (2018)	Gapidt	WSDL documents	SOAP	Service discoverability anti-pattern
Kumar and Sureka (2018)	Analysis approach	4 data sampling 5 feature ranking techniques 8 machine learning algorithms 5 different types of anti-patterns 226 real-world Web services	SOAP	God object WS (GOWS), fine-grained WS (FGWS), data WS (DWS), chatty WS (CWS) and ambiguous WS (AWS)
Velioglu and Selcuk (2017)	An attempt to achieve code-smell anti-patterns detection	Expert opinion of three software developers	Code smell	Brain method and data class code smells
Palma *et al.* (2017)	SARA	18 widely used RESTful WS	REST APIs	Amorphous URI, nonhierarchical nodes, nonpertinent documentation, contextless resource names
Sabir *et al.* (2017)	Detection approaches	60 weather and 7 finance Web services	SOAP	GOWS, FGWS, DWS, AWS, CWS, duplicated services, low cohesive operations in the same port-type, redundant port types, CRUD URIs, loosey-goosey WS

(*continued*)

Studies	Approach/tool	Setting	Architecture style	Types of anti-pattern
Brabra *et al.* (2016)	Semantic-based approach	Cloud service APIs	Cloud APIs	Missing query interface, non-compliant create, non-compliant update, non-compliant delete, non-compliant retrieve, non-compliant trigger action, non-compliant link between resources, non-compliant association of resource(s) with mixing, non-compliant dissociation of resource(s) from mixing, non-compliant URL, non-compliant request header, non-compliant response header, ignoring status code, CRUD URIs
Palma *et al.* (2015)	DOLAR	15 well-known RESTful WS	REST APIs	Contextless resource names, non-hierarchical nodes, amorphous URIs, CRUD URIs, pluralised nodes
Torkamani and Bagheri (2014)	Systematic method	Evaluated in action	SOA	So, what is new, interface bloat, shiny nickel, big bang
Palma *et al.* (2014a)	SODA-R	12 widely used REST APIs	REST APIs	Breaking self-descriptiveness, forgetting hypermedia, ignoring caching, ignoring Multipurpose Internet Mail Extensions (MIME) types, ignoring status code, misusing cookies, tunnelling through GET, tunnelling through POST
Ordiales Coscia *et al.* (2014)	An approach to avoid WSDL anti-patterns	1,664 WSDL documents	SOAP	Enclosed data model, low cohesive operations in the same port-type, redundant data models, ambiguous names, whatever types, empty messages

Table 1.

Anti-patterns in RESTful WSs may appear in their URI: a directory-structure-like of resource identifier (Rodriguez, 2008). A URI is an identifier for a specific resource, and it could be a uniform resource locator (URL) with a specific protocol such as HTTP, FTP, MAILTO and TELNET and the specific resource available to the URL. The directory-structure-like can be viewed in a tree hierarchy where the root of the URI has several nodes representing a collection of resources. A node could also have many sub-nodes. Figure 2 illustrates an example of a URI with its nodes that represent resources of a RESTful WS as described by Rodriguez (2008). In this example, a URI, http://myservice.org/discussion/topics/{topic}, contains a node named discussion, with a sub-node of topics. There are many related resources in the topic, such as gossips, technology, fashions and travel. Apart from discussion node, there is also another node call authors. A URI for this resource is http://myservice.org/authors/{author 1}.

URI is the essential component of a RESTful WS. A URI should be easy to understand and does not merely combine strings with slashes and delimiters. Brabra *et al.* (2019) argued that the URIs should have clear and representative definitions, which means the nodes should have meaningful names and relevant. Anti-patterns could also happen in the design of URIs, which are named as amorphous URIs, CRUD URI, pluralised nodes, contextless recourse names, non-hierarchical nodes and ambiguous names (Alshraiedeh and Katuk, 2019). Hirsch *et al.* (2018) and Mateos *et al.* (2015b) studied ambiguous name anti-patterns in SOAP to define the corresponding definition in RESTful WS through the SOAP description document, such as portTypes, operations and parameters. For the URIs, the corresponding nodes include resources parameters and resources path. The literature suggested different approaches to detect anti-pattern occurrences. In the case of this study, the following anti-patterns are selected because they have a direct impact on developers' understanding and using RESTful WSs.

- Ambiguous names – A URI contains nodes that are not understandable. For example, a URI www.example.com/ns/media/page?id=123 contains an ambiguous resource name, because the node "ns" has a short name and also does not have meaning. Furthermore, if the numbers or a mixture of numbers and letters but started by number are used, then, the nodes' names will be conflicted with parameters naming standard. Further, the length of the name should be between 3 and 30 characters, while other lengths are considered meaningless (De Renzis *et al.*, 2016; Petrillo *et al.*, 2018).
- Amorphous URIs – A URI contains symbols including file extensions, underscores, final trailing-slash or capital letter (Brabra *et al.*, 2019; Palma *et al.*, 2014a; Palma *et al.*, 2017; Sabir *et al.*, 2019; Palma *et al.*, 2019). For example, the URI www.example.com/NEW_Customer/photo01.jpg/ contains capital letters, underscore and file extension.

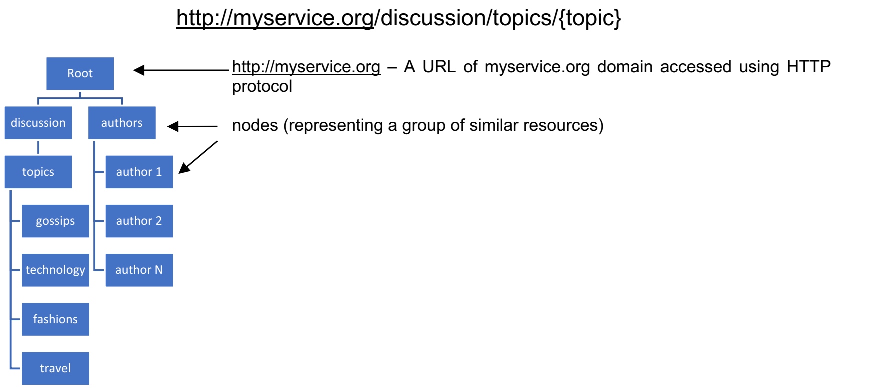

Figure 2.
An example of URI and the nodes

3. The proposed uniform resource identifiers parsing technique and algorithm for RESTful Web services anti-pattern detection

This study was conducted following the design science research process by Peffers *et al.* (2007). The process consists of six activities, namely,

(1) identifying problems;
(2) identifying solution;
(3) design the solution;
(4) demonstrate the solution;
(5) evaluation; and
(6) communicate the solution.

The problem that the study intends to address was specified in Sections 1 and 2. On the other hand, this section describes the solution and its design as well as demonstrates the implementation of the URI parsing algorithm in the anti-pattern detection technique. Then, the evaluation of the technique is described in Section 4. Finally, this paper aims at communicating the findings to the WS researchers and developers.

This remaining content of this section describes the techniques and design of the proposed algorithm for detecting the selected anti-patterns. As described earlier, anti-patterns may be found in URIs that could affect the understandability and reusability of the RESTful WSs. Therefore, addressing and tackling these anti-patterns could improve WS functionality and eventually increase usages. The detection of amorphous URIs and ambiguous names anti-patterns is accomplished using combinations of the following techniques:

- *Checking the length of the characters for naming the nodes.* The algorithm analyses the names of the URI in terms of their length. One of the main issues with the names of the URI nodes is that they tend to be too short or too long. If the length of the nodes' name is between 3 and 30 characters, then, it is considered to have an appropriate length (Petrillo *et al.*, 2017; Mateos *et al.*, 2015b; Rodriguez *et al.*, 2013). Otherwise, the name is considered as an ambiguous name. Figure 3 depicts the algorithm for examining the URI nodes length. This algorithm addresses the ambiguous names.
- *Checking the meaning of the names of the URI nodes.* The algorithm analyses the name of the URI nodes against known unrepresentative names such as param, arg, var, obj, object, foo, input, output, in# and out#, where # might be replaced by a number or nothing (Hirsch *et al.*, 2018; Mateos *et al.*, 2015b). Other characteristics that need to be tackled is to detect unrepresentative name. These names can comprise the words that do not have a dictionary entry such as meaningless, nonsense and non-standard words. In verifying the name of URI nodes, this study used variant resources (i.e. publications, websites, online

```
Inappropriate-Length-URINode
URINode← Extract URINode
for each index = 1 to Length(URINode) - 1
   if Length(Node) < 30 or Length(Node)<3 then
      Ambiguous-Node = Ambiguous-Node +1
   end if
end for
return false
```

Figure 3. Algorithm for analysing the length of the name of the URI nodes

dictionaries, etc.) to generate a collection of unrepresentative and meaningless English words, including WordNet, an online lexical database system. Figure 4 shows the proposed algorithm for examining the unrepresentative name of URI nodes (Flint *et al.*, 2017). This technique addresses both amorphous URIs and ambiguous names.

- *Checking symbols in the names of the URI nodes*. The algorithm also analyses the name of the URI nodes against symbols as illustrated in Figure 5. This technique addresses amorphous URIs. At the same time, the algorithm analyses the concurrent occurrences of ambiguous names and amorphous URI anti-patterns, as illustrated in Figure 6.

The proposed algorithm for detecting the anti-patterns in RESTful WSs has four sections as represented by Figures 1–4. In general, the algorithm parses a URI and scans it for the specified anti-patterns. This study proposed the URI parsing, a technique that analyses the URI string and spots a specific syntax or grammar that matches the given rules or corpus. The URI parsing technique is embedded in the algorithm to detect amorphous URI and ambiguous name anti-patterns found in RESTful WS. The URI parsing technique is presented again in Figure 7.

```
Unrepresentative-URINode
URINode← Extract URINode
for each index = 1 to Length(URINode) - 1
   Set1 ← Unrepresentative-Node
   Set2 ← Meaningless-Node
   Set3 ← Nonsense-Node
   If  Set1 ∪  Set2 ∪ Set3 = 1  then
      Ambiguous-Node = Ambiguous-Node +1
      Set4 ← Extract-Semantic-By-WordNet(URINode)
   end if
   If  Set4 = Ø   then
      Ambiguous-Node = Ambiguous-Node +1
   end if
end for
```

Figure 4.
Algorithm for analysing the meaning of the name in the URI nodes

```
Amorphous-Names(Request-URI)
URINode← Extract URINode
for each index = 1 to Length(URINode) - 1
   if  Node←  Symbol then
      Amorphous -Node = Amorphous -Node +1
   end if
end for
return false
```

Figure 5.
Algorithm for analysing symbols

```
Intersection-URI
URINode← Extract URINode
for each index = 1 to Length(URINode) - 1
   Set1 ← Ambiguous-Node
   Set2 ← Amorphous-Node
   If  Set1 ∩ Set2 = 1  then
      Intersection-Node = Intersection-Node +1
   end if
end for
return false
```

Figure 6.
Algorithm for detecting the concurrent occurrences of ambiguous names and amorphous URI anti-patterns

4. Evaluation and results

This section elaborates the evaluation of the proposed algorithm for detecting the anti-patterns in RESTful WSs. The section describes the evaluation procedures and the results derived from the analyses.

4.1 Evaluation procedure

The proposed algorithm described in Section 3 was evaluated using the procedure illustrated in Figure 8. The procedure consists of four phases, namely:

(1) extracting the URIs;

(2) implementing the anti-pattern detection algorithm;

(3) detecting the anti-patterns; and

(4) validating the results.

Phase 1: Extract RESTful URIs. This study collected data and formed a data set of URIs manually by crawling the online documentation of well-known RESTful WSs. The URLs of 16 online documentation of the selected RESTful WSs that made up the data set for the study are listed in Table 2. The data set was filtered to remove the redundant URIs belonged to the same RESTful WSs which resulted in unique 11,692 URIs. The “request” URIs was also removed from the data set because the amorphous URI and ambiguous name anti-patterns can be extracted from the online documentation without executing any HTTP methods and parameterised requests (Palma *et al.*, 2015).

Phase 2: Implement the detection algorithm. The principal researcher conducted a manual inspection and anti-pattern detection of each URI. It took 45 days to analyse the data set for the specified anti-patterns by applying the detection rules as specified in the algorithm for each row of the data set stored in a Microsoft Excel sheet. The principal researcher scanned the URIs row by row. As a first step, it divides the parsed URIs for tokens (i.e. parameter, a path segment or a mixture of parameters and path segments)

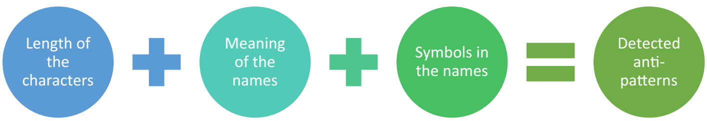

Figure 7. URI parsing technique for detecting RESTful WS anti-patterns

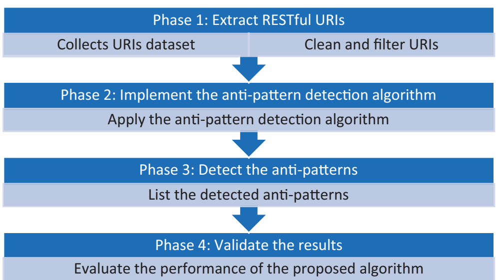

Figure 8. Evaluation procedure of the proposed algorithm

RESTful WS providers	Online documentation
BestBuy	www.developer.bestbuy.com/documentation
Bitly	www.dev.bitly.com/api.html
CharlieHarvey	www.charlieharvey.org.uk/about/api
Dropbox	www.dropbox.com/developers/core/docs
Externalip	www.api.externalip.net
Facebook	https://developers.facebook.com/docs/graph-api
GoogleBook	www.developers.google.com/books/
Instagram	www.instagram.com/developer
LinkedIn	www.developer.linkedin.com/docs
Ohloh	www.github.com/blackducksw/ohlohapi
StackExchange	www.api.stackexchange.com/docs
TeamViewer	www.integrate.teamviewer.com/en/develop/documentation
Twitter	www.dev.twitter.com/rest/public
Walmart	www.developer.walmartlabs.com/
YouTube	www.youtube.com/yt/dev/api-resources.html
Zappos	http://developer.zappos.com/docs/api-documentation

Table 2.
List of 16 RESTful WSs and their online documentations

and then extracts the resource and parameters names. This study relied on WordNet ontology – the fundamental for understanding the meaning of names and also used to explore the semantic similarity between words (Palma *et al.*, 2015).

Phase 3: Detect the anti-patterns. The algorithm was transformed into a C# code for automated detection and counting of the anti-pattern occurrence in the data set. Then, a list of detected anti-patterns from the RESTful WSs is generated. Expert review method was used to validate the results of the automated anti-pattern detection. Four professionals reviewed the list of the detected anti-patterns. Two of them were expert in linguistic, whereas the others were experts in software development. This study provided them with the definitions of ambiguous names and amorphous URI anti-patterns and the sets of all collected URIs from Twitter, Facebook, YouTube, Dropbox and BestBuy in Microsoft Excel sheet. The experts manually tested the subset URIs to identify the true-positive and false-negative in defining ground truth for the analysed URIs. The professionals were asked to set 1 when an anti-pattern detected and 0 for otherwise. Then, 250 URIs from professionally tested subset were randomly selected to measure the overall accuracy of the manually detected anti-patterns.

Phase 4: Validate the results. The accuracy of the proposed algorithm was validated using three performance measures, as shown in equations (1)-(3), respectively:

$$Recall = \frac{True\ \ detected\ \ anti - patterns}{All\ \ exsisting\ \ true\ \ anti - patterns} \tag{1}$$

$$Precision = \frac{True\ \ detected\ \ anti - patterns}{All\ \ detected\ \ anti - patterns} \tag{2}$$

$$F - measure = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{3}$$

The recall represents the ratio between the true-detected anti-patterns and all existing true anti-patterns, whereas precision represents the ratio between the true-detected anti-patterns and all detected anti-patterns (Palma *et al.*, 2015; Sabir *et al.*, 2019).

4.2 Analysis and results

The data set contained a total of unique 11,692 URIs from the 16 RESTful WS providers. The specific number of URIs for each provider is listed in Table 3. Dropbox and Facebook had the highest and lowest number of URIs, respectively, with 2,607 and 3. The nodes from each URI were separated and counted, as shown in the third column of Table 3. The URIs generated 39,541 number of nodes. In average, each URI had 2.75 nodes with 1.218 of standard deviation. Twitter generated 5.4 nodes on average from each URI for the highest range, whereas Facebook generated one node representing the lowest range. The details for all collected URIs, resources' paths and nodes for the 16 RESTful WS providers are available on GitHub at https://github.com/Fuad81/rest-api

The manual anti-pattern detection processes discovered the ambiguous names, amorphous URIs and their concurrent anti-pattern appearance from the data set, as shown in Table 4. The analysis suggested a total of 6,787 intersection anti-patterns. The results of this study reveal that 97% of the Dropbox RESTful URIs contained amorphous URI anti-patterns, whereas 99% of Walmart URIs contained ambiguous name anti-patterns, the highest occurrences. Ambiguous name anti-pattern was not found in Facebook RESTful WS, whereas 24% of amorphous URI anti-patterns were found on Instagram, the lowest number of anti-pattern occurrences. The highest number of concurrent anti-pattern appearance was found in Walmart and Dropbox RESTful WSs with 92% and 91%, respectively. No concurrent anti-pattern appearance was found on Facebook, whereas 2% of it was found in StackExchange, which contributed to the lowest percentage. The bar chart in Figure 9 shows the percentage of anti-pattern occurrences.

The last part of the analysis involved validating the accuracy of the proposed technique and algorithm. The automated detection programmed in C# code was run, and the results derived from the programme were compared with the results of the manual detected anti-patterns. Table 5 shows the accuracy measures that cover recall, precision and F-measure. The results demonstrate that the mean of precision, recall and F-measure was 82.30%, 87.86% and 84.93%,

Providers of RESTful WSs	Total no. of URIs	Total no. of URI nodes
BestBuy	502	1,096
Bitly	33	38
CharlieHarvey	2,091	4,699
Dropbox	2,607	8,069
Externalip	29	49
Facebook	3	3
GoogleBook	1,656	7,779
Instagram	33	82
LinkedIn	1,676	6,838
Ohloh	492	1,144
StackExchange	231	506
TeamViewer	212	726
Twitter	979	5,298
Walmart	609	2,120
YouTube	513	1,030
Zappos	26	64
Total	11,692	39,541

Table 3. Number of URIs and their associated nodes

RESTful WS	Ambiguous names anti-pattern	Amorphous URI anti-pattern	Ambiguous names ∩ Amorphous URI
BestBuy	183	232	79
Bitly	6	16	3
CharlieHarvey	1,385	661	547
Dropbox	2,377	2,520	2,374
Externalip	11	35	11
Facebook	0	1	0
GoogleBook	658	1,251	626
Instagram	6	8	5
LinkedIn	1,399	1,454	1,297
Ohloh	192	220	107
StackExchange	6	179	5
TeamViewer	206	151	151
Twitter	939	889	864
Walmart	605	562	561
YouTube	433	176	152
Zappos	5	19	5

Table 4.
Anti-patterns occurrence in the selected RESTful WS

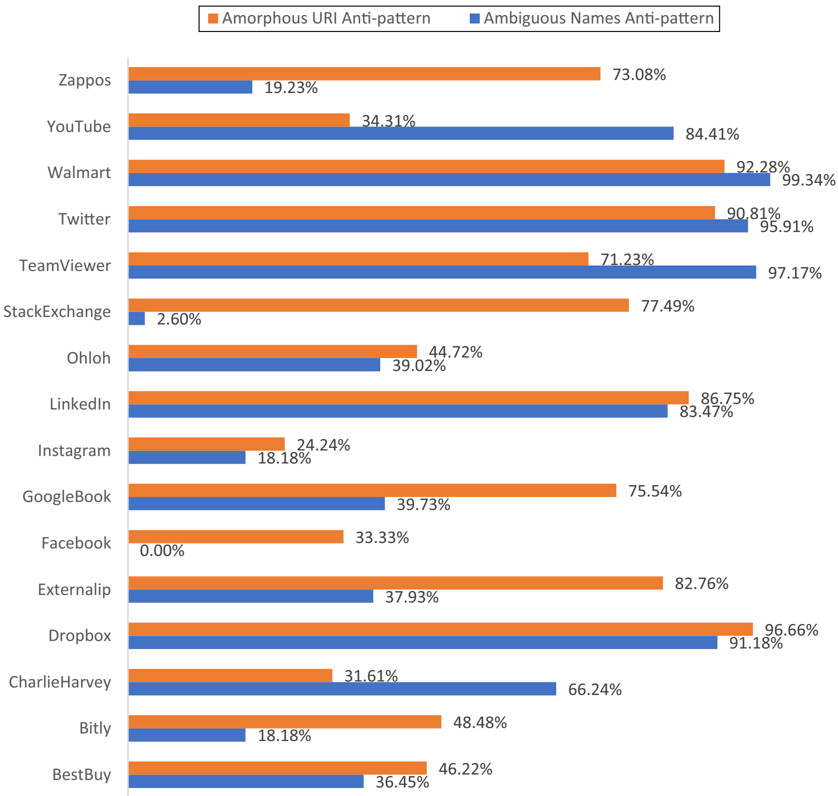

Figure 9.
Percentage of anti-pattern occurrences

Table 5. Accuracy measures of the proposed technique and algorithm

RESTful WS	Ambiguous name anti-pattern	Amorphous URI anti-pattern	Recall Ambiguous names (%)	Recall Amorphous URI (%)	Precision Ambiguous names (%)	Precision Amorphous URI (%)	F-measure Ambiguous names (%)	F-measure Amorphous URI (%)
BestBuy	183	232	82.51	80.17	91.52	88.57	86.78	84.16
Bitly	6	16	83.33	87.50	100.00	87.50	90.91	87.50
CharlieHarvey	1,385	661	85.20	79.58	97.93	88.26	91.12	83.69
Dropbox	2,377	2,520	82.04	82.98	90.91	89.13	86.25	85.94
Externalip	11	24	81.82	83.33	81.82	83.33	81.82	83.33
Facebook	0	1	100.00	100.00	100.00	100.00	100.00	100.00
GoogleBook	658	1,251	77.66	82.25	85.31	93.38	81.30	87.46
Instagram	6	8	83.33	75.00	83.33	75.00	83.33	75.00
LinkedIn	1,399	1,454	83.06	81.98	90.43	86.06	86.59	83.97
Ohloh	192	220	80.21	82.27	85.56	89.60	82.80	85.78
StackExchange	6	179	66.67	85.47	66.67	90.53	66.67	87.93
TeamViewer	206	151	82.04	82.78	94.41	86.21	87.79	84.46
Twitter	939	889	85.84	84.93	97.70	92.07	91.38	88.36
Walmart	605	562	82.15	80.96	87.96	90.64	84.96	85.53
YouTube	433	176	83.14	81.25	87.38	86.14	85.21	83.63
Zappos	5	19	60.00	84.21	60.00	84.21	60.00	84.21
Average			81.19	83.42	87.56	88.17	84.18	85.68

respectively, for the proposed parsing technique and the algorithm. The accuracy measures suggested that the proposed technique and algorithm is acceptable for a valid prediction of amorphous URI and ambiguous name anti-patterns in the URIs of RESTful WSs.

5. Conclusion and future work

Many RESTful WSs suffered from amorphous URI and ambiguous name anti-patterns which caused difficulty for users (or programmes) in understanding and reusing them. Consequently, it resulted in underused services which diminish their sustainability. Anti-patterns problem happened when developers practised weak design principles for the RESTful WSs. Amorphous URI and ambiguous name anti-patterns can be detected by experienced software developers manually; however, it is a time-consuming task. Therefore, an automated detection technique and algorithm can have a practical contribution in detecting the anti-patterns which help developers at an early stage of the RESTful WSs development. This study proposed a URI parsing technique that is embedded in an algorithm to detect the two anti-patterns. Although only two anti-patterns were studied, the contribution is significant to assist developers in detecting anti-patterns and improving the URIs for better understanding. This study also constructed a data set of URIs derived from 16 RESTful WS providers that could be useful for other researchers who are studying the same topic. In future, this study aims to extend the technique to extract more types of anti-patterns, and also extract anti-patterns from different types of RESTful documentation such as Web application description and swagger. Another potential work is applying artificial intelligence techniques to detect the anti-patterns from the developed data set.

References

Abidi, S., Fakhri, M., Essafi, M. and Ben Ghazela, H. (2019), "A comprehensive framework for evaluating web services composition methods", *International Journal of Web Information Systems*, pp. 324-345, doi: 10.1108/IJWIS-04-2018-0027.

Aghajani, E., Nagy, C., Bavota, G. and Lanza, M. (2018), "A large-scale empirical study on linguistic antipatterns affecting APIs", in *Proceedings – 2018 IEEE International Conference on Software Maintenance and Evolution, ICSME 2018*, pp. 25-35, doi: 10.1109/ICSME.2018.00012.

Ahmad, R., Hussain, A. and Baharom, F. (2017), "Embedding the concept of service-oriented architecture into software sustainability evaluation model", *in AIP Conference Proceedings*, p. 20023, doi: 10.1063/1.5005356.

Alshraiedeh, F. and Katuk, N. (2016), "Enrichment of data type specification for web service", in *The 3rd Innovation and Analytics Conference and Exhibition (IACE) 2016, 31 October – 1 November 2016*, available at: https://sites.google.com/site/sqsiace/previous-conferences/iace-2016-proceedings

Alshraiedeh, F., Hanna, S. and Alazaidah, R. (2015), "An approach to extend WSDL-based data types specification to enhance web services understandability", *International Journal of Advanced Computer Science and Applications*, Vol. 6 No. 3, pp. 88-98, doi: 10.14569/IJACSA.2015.060314.

Alshraiedeh, F. and Katuk, N. (2019), "SOAP and RESTful web service anti-patterns: a scoping review", *International Journal of Advanced Trends in Computer Science and Engineering*, Vol. 8 No. 5, pp. 1831-1840, doi: 10.30534/ijatcse/2019/05852019.

Aradea, A., Supriana, I. and Surendro, K. (2020), "Self-adaptive model based on goal-oriented requirements engineering for handling service variability", *Journal of Information and Communication Technology*, Vol. 19 No. 2, pp. 225-250, doi: 10.32890/jict2020.19.2.4.

Atlidakis, V., Godefroid, P. and Polishchuk, M. (2019), "RESTler: Stateful REST API fuzzing", in *Proceedings – International Conference on Software Engineering*, pp. 748-758, doi: 10.1109/ICSE.2019.00083.

Boukharata, S., Ouni, A., Kessentini, M., Bouktif, S. and Wang, H. (2019), "Improving web service interfaces modularity using multi-objective optimization", *Automated Software Engineering*, Vol. 26 No. 2, pp. 275-312, doi: 10.1007/s10515-019-00256-4.

Brabra, H., Mtibaa, A., Petrillo, F., Merle, P., Sliman, L., Moha, N., Gaaloul, W., Guéhéneuc, Y.G., Benatallah, B. and Gargouri, F. (2019), "On semantic detection of cloud API (anti)patterns", *Information and Software Technology*, Vol. 107, pp. 65-82, doi: 10.1016/j.infsof.2018.10.012.

Brabra, H., Mtibaa, A., Sliman, L., Gaaloul, W., Benatallah, B. and Gargouri, F. (2016), "Detecting cloud (anti)patterns: OCCI perspective", in *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, pp. 202-218, doi: 10.1007/978-3-319-46295-0_13.

De Renzis, A., Garriga, M., Flores, A., Cechich, A. and Zunino, A. (2016), "Case-based reasoning for web service discovery and selection", *Electronic Notes in Theoretical Computer Science*, Vol. 321, pp. 89-112, doi: 10.1016/j.entcs.2016.02.006.

De Renzis, A., Garriga, M., Flores, A., Cechich, A., Mateos, C. and Zunino, A. (2017), "A domain-independent readability metric for web service descriptions", *Computer Standards and Interfaces*, Vol. 50, pp. 124-141, doi: 10.1016/j.csi.2016.09.005.

Flint, E., Ford, E., Thomas, O., Caines, A. and Buttery, P. (2017), "A text normalisation system for non-standard English words", in *Proceedings of the 3rd Workshop on Noisy User-generated Text*, pp. 107-115, doi: 10.18653/v1/W17-4414.

Fokaefs, M. and Stroulia, E. (2015), "Using WADL specifications to develop and maintain REST client applications", in *Proceedings – 2015 IEEE International Conference on Web Services, ICWS 2015*, pp. 81-88, doi: 10.1109/ICWS.2015.21.

Hamzah, M.H.I., Baharom, F. and Mohd, H. (2019), "An exploratory study for investigating the issues and current practices of service-oriented architecture adoption", *Journal of Information and Communication Technology*, Vol. 18 No. 3, pp. 273-304, doi: 10.32890/jict2019.18.3.3.

Hamzah, M.H.I., Baharom, F., Mohd, H. and Omar, M.H. (2018), "A construction of service-oriented architecture adoption maturity levels using adoption of innovation concept and CMMI", *Journal of Telecommunication, Electronic and Computer Engineering (JTEC)*, Vol. 10 Nos 2/4, pp. 23-27.

Hirsch, M., Rodriguez, A., Rodriguez, J.M., Mateos, C. and Zunino, A. (2018), "Spotting and removing WSDL anti-pattern root causes in code-first web services using NLP techniques: a thorough validation of impact on service discoverability", *Computer Standards and Interfaces*, Vol. 56, pp. 116-133, doi: 10.1016/j.csi.2017.09.010.

Huang, C.Y. and Wu, C.H. (2016), "A web service protocol realizing interoperable internet of things tasking capability", *Sensors*, Vol. 16 No. 9, p. 1395, doi: 10.3390/s16091395.

Kallab, L., Chbeir, R. and Mrissa, M. (2019), "Automatic K-resources discovery for hybrid web connected environments", in *Proceedings – 2019 IEEE International Conference on Web Services, ICWS 2019 – Part of the 2019 IEEE World Congress on Services*, pp. 146-153, doi: 10.1109/ICWS.2019.00034.

Kalyani, G.K. (2018), "Search based web service and business process anti pattern detection", *International Journal for Research in Engineering Application and Management*, Vol. 4 No. 3, pp. 558-561, doi: 10.18231/2454-9150.2018.0380.

Kehagias, D., Mavridou, E., Giannoutakis, K.M., Tzovaras, D. and Hassapis, G. (2018), "Automatic categorization of web service elements", *International Journal of Web Information Systems*, Vol. 14 No. 2, pp. 233-258, doi: 10.1108/IJWIS-08-2017-0059.

Kumar, L. and Sureka, A. (2018), "An empirical analysis on web service anti-pattern detection using a machine learning framework", in *Proceedings – International Computer Software and Applications Conference*, pp. 2-11, doi: 10.1109/COMPSAC.2018.00010.

Lizarralde, I., Mateos, C., Rodriguez, J.M. and Zunino, A. (2019), "Exploiting named entity recognition for improving syntactic-based web service discovery", *Journal of Information Science*, Vol. 45 No. 3, pp. 398-415, doi: 10.1177/0165551518793321.

Ma, S.P., Lin, H.J., Lan, C.W., Lee, W.T. and Hsu, M.J. (2017), "Real-world RESTful service composition: a transformation-annotation-discovery approach", in *Proceedings – 2017 IEEE 10th International Conference on Service-Oriented Computing and Applications, SOCA 2017*, pp. 1-8, doi: 10.1109/SOCA.2017.9.

Ma, S., Chen, Y., Syu, Y., Lin, H. and Fanjiang, Y. (2018), "Test-Oriented RESTful service discovery with semantic interface compatibility", *IEEE Transactions on Services Computing*, pp. 1-1, doi: 10.1109/TSC.2018.2871133.

Mateos, C., Crasso, M., Zunino, A. and Ordiales Coscia, J.L. (2015a), "A stitch in time saves nine: early improving code-first web services discoverability", *International Journal of Cooperative Information Systems*, Vol. 24 No. 2, p. 1550004, doi: 10.1142/S0218843015500045.

Mateos, C., Rodriguez, J.M. and Zunino, A. (2015b), "A tool to improve code-first web services discoverability through text mining techniques", *Software: Practice and Experience*, Vol. 45 No. 7, pp. 925-948, doi: 10.1002/spe.2268.

Nayrolles, M., Moha, N. and Valtchev, P. (2013), "Improving SOA antipatterns detection in service based systems by mining execution traces", in *Proceedings – Working Conference on Reverse Engineering, WCRE*, pp. 321-330, doi: 10.1109/WCRE.2013.6671307.

Ordiales Coscia, J.L., Mateos, C., Crasso, M. and Zunino, A. (2014), "Refactoring code-first web services for early avoiding WSDL anti-patterns: approach and comprehensive assessment", *Science of Computer Programming*, Vol. 89, pp. 374-407, doi: 10.1016/j.scico.2014.03.015.

Ouni, A., Salem, Z., Inoue, K. and Soui, M. (2016), "SIM: an automated approach to improve web service interface modularization", in *Proceedings – 2016 IEEE International Conference on Web Services, ICWS 2016*, pp. 91-98, doi: 10.1109/ICWS.2016.20.

Ouni, A., Wang, H., Kessentini, M., Bouktif, S. and Inoue, K. (2019), "A hybrid approach for improving the design quality of web service interfaces", *ACM Transactions on Internet Technology*, Vol. 19 No. 1, pp. 1-24, doi: 10.1145/3226593.

Palma, F. (2015), "Unifying service oriented technologies for the specification and detection of their antipatterns", PhD, École Polytechnique de Montréal.

Palma, F., Dubois, J., Moha, N. and Guéhéneuc, Y.G. (2014a), "Detection of REST patterns and antipatterns: a heuristics-based approach", in *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, pp. 230-244, doi: 10.1007/978-3-662-45391-9_16.

Palma, F., Gonzalez-Huerta, J., Founi, M., Moha, N., Tremblay, G. and Guéhéneuc, Y.G. (2017), "Semantic analysis of RESTful APIs for the detection of linguistic patterns and antipatterns", *International Journal of Cooperative Information Systems*, Vol. 26 No. 2, pp. 1742001-1-1742001-37, doi: 10.1142/S0218843017420011.

Palma, F., Gonzalez-Huerta, J., Moha, N., Guéhéneuc, Y.G. and Tremblay, G. (2015), "Are RESTful APIs well-designed? Detection of their linguistic (anti)patterns", in *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, pp. 171-187, doi: 10.1007/978-3-662-48616-0_11.

Palma, F., Moha, N. and Gueheneuc, Y.G. (2019), "UniDoSA: the unified specification and detection of service antipatterns", *IEEE Transactions on Software Engineering*, Vol. 45 No. 10, pp. 1024-1053, doi: 10.1109/TSE.2018.2819180.

Palma, F., Moha, N., Tremblay, G. and Guéhéneuc, Y.G. (2014b), "Specification and detection of SOA antipatterns in web services", in *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, pp. 58-73, doi: 10.1007/978-3-319-09970-5_6.

Peffers, K., Tuunanen, T., Rothenberger, M.A. and Chatterjee, S. (2007), "A design science research methodology for information systems research", *Journal of Management Information Systems*, Vol. 24 No. 3, pp. 45-77, doi: 10.2753/MIS0742-1222240302.

Petrillo, F., Merle, P., Moha, N. and Guéhéneuc, Y.-G. (2017), "Towards a REST cloud computing lexicon", in *7th International Conference on Cloud Computing and Services Science, CLOSER 2017*, pp. 376-383. 10.5220/0006281203760383.

Petrillo, F., Merle, P., Palma, F., Moha, N. and Guéhéneuc, Y.G. (2018), "A lexical and semantical analysis on REST cloud computing APIs", *In: Communications in Computer and Information Science*, pp. 308-332.

Rai, G.N., Gangadharan, G.R. and Padmanabhan, V. (2015), "Algebraic modeling and verification of web service composition", *Procedia Computer Science*, Vol. 52, pp. 675-679, doi: 10.1016/j.procs.2015.05.072.

Rodriguez, A. (2008), "Restful web services: the basics", *IBM developerWorks*, Vol. 33, p. 18.

Rodriguez, J.M., Crasso, M. and Zunino, A. (2013), "An approach for web service discoverability anti-pattern detection for journal of web engineering", *Journal of Web Engineering*, Vol. 12 Nos 1/2, pp. 131-158.

Rodriguez, J.M., Mateos, C. and Zunino, A. (2015), "Assisting developers to build high-quality code-first web service APIs", *Journal of Web Engineering*, Vol. 14 Nos 3/4, pp. 251-285, available at: https://developer.ibm.com/technologies/web-development/articles/ws-restful/

Ruokonen, A., Wu, Z. and Lu, R. (2016), "Describing mobile devices as RESTful services for the end-users", in *Proceedings – 2016 IEEE International Conference on Mobile Services, MS 2016*, pp. 127-134, doi: 10.1109/MobServ.2016.27.

Sabir, F., Palma, F., Rasool, G., Guéhéneuc, Y.G. and Moha, N. (2019), "A systematic literature review on the detection of smells and their evolution in object-oriented and service-oriented systems", *Software: Practice and Experience*, Vol. 49 No. 1, pp. 3-39, doi: 10.1002/spe.2639.

Sabir, F., Rasool, G. and Yousaf, M. (2017), "A lightweight approach for specification and detection of soap anti-patterns", *International Journal of Advanced Computer Science and Applications*, Vol. 8 No. 5, pp. 455-467, doi: 10.14569/IJACSA.2017.080555.

Saluja, S. and Batra, U. (2019), "Assessing quality by anti-pattern detection in web services", *in Proceedings of International Conference on Sustainable Computing in Science, Technology and Management (SUSCOM), Amity University Rajasthan, Jaipur-India*, pp. 47-52, doi: 10.2139/ssrn.3350876.

Stamelos, I. (2010), "Software project management anti-patterns", *Journal of Systems and Software*, Vol. 83 No. 1, pp. 52-59, doi: 10.1016/j.jss.2009.09.016.

Torkamani, M.A. and Bagheri, H. (2014), "A systematic method for identification of anti-patterns in service oriented system development", *International Journal of Electrical and Computer Engineering (Ijece)*, Vol. 4 No. 1, pp. 16-23, doi: 10.11591/ijece.v4i1.4097.

Velioglu, S. and Selcuk, Y.E. (2017), "An automated code smell and anti-pattern detection approach", in *Proceedings – 2017 15th IEEE/ACIS International Conference on Software Engineering Research, Management and Applications, SERA 2017*, pp. 271-275, doi: 10.1109/SERA.2017.7965737.

Xie, F., Wang, J., Xiong, R., Zhang, N., Ma, Y. and He, K. (2019), "An integrated service recommendation approach for service-based system development", *Expert Systems with Applications*, Vol. 123, pp. 178-194, doi: 10.1016/j.eswa.2019.01.025.

Xue, Y., Zhang, C. and Ji, Y. (2015), "RESTful web service matching based on WADL", in *Proceedings – 2015 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery, CyberC 2015*, pp. 364-371, doi: 10.1109/CyberC.2015.89.

Corresponding author
Norliza Katuk can be contacted at: k.norliza@uum.edu.my