2021 International Conference on Intelligent Computing, Automation and Systems (ICICAS) | 978-1-6654-2810-1/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICICAS53977.2021.00059

A Review of Bad Smells in Cloud-based Applications and Microservices

Di Guo
Shanghai Normal University
Department of Computer Science and Technology
Shanghai 200234, China
gdleroy@163.com

Haitao Wu
Shanghai Normal University
Department of Computer Science and Technology
Shanghai 200234, China
ht.wu@163.com

Abstract—**Cloud- and Microservice-based applications are continuously gaining attention in the field of practical software engineering as well as in academic research, but the bad practices exist in such applications have not yet been systematically discussed and analyzed. Moreover, the impact of bad practices to actual systems and code quality indicators, *e.g.* reliability, testability, maintainability, are remaining unknown, making systematic review an important task to fulfill. The author uses Code Smell, *i.e.* the sub-optimal implementation and design choices, as an aspect to analyze empirically the bad practice of Cloud and Microservice based applications. This paper investigated 33 infrastructure configuration smells and 34 microservice smells through a Systematic Literature Review (SLR), and provided 2 appendixes concerning the above-mentioned 2 kinds of smells. Additionally, this paper also predicts the future development of this field according to actual works done and the gaps to be filled. Future research in related area can rely on this paper as a reference.**

Keywords—bad smell, cloud computing, microservice, architecture, infrastructure

I. Introduction

Code Smell is defined as symptoms of bad implementation and design choices [1]. Code Smell is introduced by software developers, and it will remain in the code for a long time with constantly growing intensity [1]. The cause of smell might be technical [1] or social [2], including high work load, lacking domain knowledge, or shortage in flexibility of development communities.

Code Smell is known for its negative effect to software system and its contribution to software degradation [1]. Research shows that Code Smell are indicators of low testability [3], high error-proneness [3], high change-proneness [4], potential bugs and defect [5] of code components being affected by smells. Researchers and practitioners presented empirical research results, and developed tools to evaluate Code Smell intensity [6], and prioritize its order [6] while perform refactoring, providing automatic [1] and semi-automatic [7] approaches.

The early definition of Code Smell by Fowler suggested 23 anti-patterns in software production code. As researchers continuously stretch the scope of bad development practice, Code Smell is also extended to different context, revealing sub-optimal practice in both related fields, including topics such as Requirement Engineering [8], Deep Learning [9], Game Development [10], Cloud Computing [11] and Microservice [12].

Cloud-based Applications provide services in a way of SaaS (Software as a Service), which are both developed and executed in a way that cloud platforms demanded and operated [11]. Such applications are gaining attention due to their lower cost, high availability, and quality of service for the users [11].

Microservices are relatively small and autonomous services that work together. Such services are modeled around a business capability, and have a single and clearly defined purpose [12]. Applying Microservice as service architecture is a major choice by cloud-based application developers [11].

Cloud-based Applications and Microservices are not immune from bad practices and smells. There exists both statement-wide and architecture-wide design and implementation flaws in such applications. These practical problems have already been discussed widely in well-known developer social sites such as Stack Overflow.

This paper provides a Systematic Literature Review (SLR) of Bad Smells in Cloud-based Applications and Microservices. The main contribution of this work includes: (1) Investigates and verifies 33 cloud-related infrastructure and configuration smells, as well as 34 Microservice smells; (2) Divides literatures into catalogues by multiple aspects of latest progress in general Code Smell research, *e.g.* detailed smell granularities, impact analysis, refactoring suggestions; (3) Makes predictions about future trends to help researchers better focus on filling the huge gap between research and practice.

The rest of the paper is organized as follows: in the second section, this paper states the scope of literatures and build up the settings of research questions; next, the results of SLR is presented, followed by the discussions about related works. Finally, this paper makes a conclusion about the work and its future development.

II. Scope and Research Design

A. Scope of the SLR

The scope of the SLR is an intersection of 3 academic topics, including Bad Smell, Cloud Computing, and Microservice. The selection task of the SLR includes filtering and snowballing phase. The purpose of filtering is to find out initially the most relevant papers that fit with the SLR's scope, while the snowballing phase is to recurrently search relevant papers in the reference section.

The author collects 560 literatures of Code Smells and related Bad Smells using title and abstract keyword filter from DBLP, IEEE Xplore, Engineering Village and Science Direct from 2002 to 2021, including early access papers not being published. In DBLP, the keyword was "smell" as it only collects Computer Science related papers. In other databases, the author applied the term "code smell" or "bad smell". To clarify, the term "anti-pattern" is not included in the scope as it is an aged general concept, and it is not clearly defined and has the problem of low acceptance among developers [13]. The author also excluded general service smell paper as "service" could be an alias of "monolithic web applications", thus general Code Smells would fit this scenario, which may impact the specialty of this paper. However, during the snowballing phase after filtering papers in databases, the author includes papers having above-mentioned terms because they tend to focus on smell-related topics.

Due to the problem of time and workload, the author did not investigate other well-known databases such as ACM and Springer, but Computer Science literatures of aforementioned databases in high quality in terms of Impact Factor and reputation were already indexed in DBLP and Engineering Village. Among these literatures, the author handpicked 10 related conference and journal paper. Afterwards, the author performed a snow-balling process, *i.e.* searching for related literatures in papers' reference. As a result, 10 papers and added 13 more paper to the scope. In conclusion, this review includes 23 papers in terms of Cloud Computing and Microservice.

B. Research Questions

This paper aims at providing a systematic investigation and insights of future development for researchers and practitioners about architecture and Code Smells in the field of Cloud Computing and Microservice. Thus, the author poses research questions as follows:

RQ1 Smell Definitions and Catalogues: *What kind of smells related to cloud applications and Microservices are described and detected in scientific literature?*

Smells in cloud applications and Microservices can be classified into different catalogues due to their nature such as granularity and type of related code component. A general catalogue can split the smell detection and impact analysis into different research directions, and guides further works to be more specific and focused rather than dispersing the authors' focus to multiple scattered and tangled tasks.

RQ2 Impact and Refactoring: *Are the impact and refactoring approach of such smells discussed in literatures?*

Code Smell related works follow generally 3 steps [14], *i.e.* (a) smell detection, (b) impact analysis, (c) smell refactoring, to accomplish the goal of improving code quality in terms of maintainability, reliability, testability, etc.

The research of cloud- and service-based smell is an emerging novel topic. As the detection methods were already provided in most of the literatures, it would be constructive to investigate if further steps are already being taken. This research question is proposed to access the livingness of this field and provide basic facts for answering RQ3.

RQ3 Future Development: *What would the future development of cloud- and service-related smell research be like?*

To answer this question, the author must investigate thoroughly not only the existing literature about cloud and service smells, but also the pattern and trend of general Code Smell literatures. Afterwards, the author would draw his own conclusion about this RQ accordingly, and provide suggestions to practitioners and researchers.

III. RESULTS

The author captures 2 major directions of cloud and Microservice smell, including infrastructure and configuration smell related closely to DevOps task, the architectural, DevOps and practical smells exist in Microservice. Additionally, a literature accessing the impact of automatic smell refactoring to application performance is also introduced.

A. Infrastructure and Configuration Smell

Infrastructure as Code (IaC) is the practice of specifying computing system configurations through code, and managing them through traditional software engineering methods. The wide adoption of configuration management and increasing size and complexity of the associated code, prompt for assessing, maintaining, and improving the configuration code's quality.

For example, a server cluster that contains numerous nodes with different hardware configurations and different software package requirements can be specified using configuration management languages such as Puppet, Chef, or Ansible and deployed automatically without human intervention. Service-based cloud computing are in great need of IaC scripts and includes various IaC technologies. However, as the scale of scripts grows, the code quality of scripts draws little attention, and it has already caused some accidents in production [15].

Thus, researchers focus on IaC script smells and produced a series of works. As is shown in Table I, the author lists an overview of literatures about IaC smell and captures some basic pattern of these literatures including technology, the presence of implementation and design smells, detection method, the presence of distribution and co-occurrence analysis, and the existence of refactoring suggestion and impact analysis.

The author has observed 5 sets of different academic groups and authors focusing on research in this field, while the source of these works also remain scattered, including premium journals and conferences in software engineering such as *ICSE*, *IST* and *MSR.*

B. Microservice Smell

Microservices are relatively small and autonomous services that work together, which are modeled around a business capability, and have a single and clearly defined purpose. This kind of architecture is currently enjoying increasing popularity and diffusion in industrial environments.

As the scale of the whole application grows, Microservice-based applications suffer from not only technical implementation problem but also high-level structural problem in architecture and endpoints.

TABLE I. AN OVERVIEW OF LITERATURES ABOUT IAC SMELL. IMP IS FOR IMPLEMENTATION, AND DES IS FOR DESIGN. ● REPRESENTS EXISTENCE, AND – STANDS FOR NO SUCH CONTENT.

Literature	IaC Technology	IMP Smells	DES Smells	Detection Method	Distribution	Co-occurrence	Refactoring	Impact
MSR 2016 [11]	Puppet	●	●	Developed Puppeteer, Metric and rule-based	●	●	Best Practice Partially Provided for IMP Smells	Design Smell and the size of configuration project, etc.
QUATIC 2018 [16]	Chef	●	●	Extended Foodcritic, Rule-based	●	–	Best Practice Provided	–
ICSE 2019 [17]	Puppet	● (Security)	–	Developed SLIC, Rule-based	●	–	Best Practice Provided	Practitioners' perception
WIMS 2020 [18]	TOSCA	●	–	Developed tosca-parser, Rule-based	–	–	Best Practice Provided	–
IST 2019 [19]	Puppet	●	●	Developed,Machine Learning	–	–	–	Defect Prediction
ACCESS 2019 [20], 2020 [21] & COMPSAC 2019 [22]	Docker	●	●	Applied, Hadolint, Rule-based & Dynamic Analysis	●	●	Suggestions Provided	Project characteristics and Smell

The multi-directional communication protocol in need of low interaction latency and the heterogeneous structure of services also caused problem such as the increase in both network transfer load and cognitive complexity.

Thus, Microservice Smells are captured by researchers to describe architectural, practical and DevOps bad practices in the microservice lifecycle including development and maintenance. As Bogner *et al.* [32] has already made a complete SLR of service-based antipatterns, the author focus on assessment and re-evaluation of the results.Table II lists the overview of literatures introducing Microservice smells for the first time. The author did not include all literatures about microservice, as such type of SLR has already been made by [32], and the main perception of this paper is to pick literatures having obvious impact and representative in this field.

C. Other Literatures

The author also captured a literature discussing whether tool-based automatic refactoring of Code Smells will impact the resource utilization of Cloud Software [34] by evaluating the consumption of CPU and memory. The result suggest that refactoring Code Smell automatically will generally increase resource usage, and for some cases (e.g. refactoring God Class), the usage could be extremely high.

IV. DISCUSSION

In this section, the author discusses the 3 posed research question to reveal the nature and development of the fields concerned by this SLR.

A. RQ1: Smell Definition and Catalogues

What kind of smells related to cloud applications and microservices are described and detected in scientific literature?

In the field of Microservices, researches prefer to discuss architectural smell topics rather than other smells including practical and DevOps smell. The cause of this problem might be the latter 2 kinds of smells are harder to detect and quantify, and there may be a lack of acceptance by developers.

As for detection method, researches of Microservices are less interested in describing the actual quantification method in rules or metrics, but they refer to describe such problems in natural language. This might be a consequence of high abstraction of Microservices' structure, detecting design problems automatically may be a challenging task.

However, the works of Nayrolles, Moha, Palma *et al.* [27, 28] provided the research community with a series of metric- and rule-based quantification method, making further impact analysis possible. As the members of those works come from the PTIDEJ Team that developed the *de-facto* baseline tool *DECOR* for Code Smell detection, they followed the existing pattern of *DECOR* and developed a detection toolset and definition framework using *RULE CARD*-based approach called SODA-W. Other works [33] also applied SBST and genetic algorithms to derive rules and provide thresholds for metrics, but such works draw less attention even if they claim that they have achieved better performance. Like many sub-domains of Code Smell, Microservice smell researches are still in a short of empirical research and impact analysis.

To the author's great astonishment, even the distribution of smells is rarely discussed. From the perspective of refactoring, researchers are likely to provide suggestions for refactoring, but they are not thinking of providing an automatic process.

In terms of IaC scripts, researchers focus on various technologies, including Puppet, Chef, Docker, *etc.*, which is essential as some smells are technology-dependent. Discussions on implementation smells exist in every literature, while two-third of the literatures mentioned design smells. Most of the paper developed or extended detection tools, but there exist some exceptions.

Sharma *et al.* 's work [11] is the most cited one providingsolid detection and discussion about configuration code smells. They also provided a fine-grained dataset extracted from GHTorrent, i.e. the mirror of GitHub Repositories in the form of a torrent, which benefitted further researches [17-20]. However, the authors did not mention how they evaluate the precision of their self-developed tool, which made it an unideal baseline for detection works.

TABLE II. AN OVERVIEW OF LITERATURES INTRODUCING NEW MICROSERVICE SMELL. ARC IS FOR ARCHITECTURAL, AND PRA IS FOR PRACTICAL. ● REPRESENTS EXISTENCE, AND – STANDS FOR NO SUCH CONTENT.

Literature	Technological Term	ARC Smells	PRA Smells	DevOps Smells	Detection Method	Distribution	Refactoring	Impact
Rotem-Gal-Oz, *Manuscript* 2002 [23]	SOA	●	–	–	Descriptions	–	Best Practice Provided	–
ICSEA 2007 [24]	SOA	●	●	–	Descriptions	–	Best Practice Provided	–
QoSA 2009 [25]	Architectural	●	–	–	Descriptions	●	Best Practice Provided	Examples from Industrial
ComputationWorld 2009 [26]	SOA	●	●	●	Descriptions	–	Best Practice Provided	–
WCRE 2013 [27], *TSE 2018* [28]	SOA	●	–	–	Developed SODA-W, Rule-based & Dynamic Analysis	●	–	–
PPAP 2015 [29]	Service	●	–	–	Developed Rule-card, Rule-based & Dynamic Analysis	–	–	–
Richards, *Manuscript* 2016 [30]	Microservice	–	●	–	Descriptions	–	Best Practice Provided	–
IWoR 2018 [31]	Microservice	●	●	●	Descriptions	–	Best Practice Provided	–
IEEE Software 2018 [12]	Microservice	●	●	●	Descriptions	●	Best Practice Provided	Questionnaire and Empirical Research

Rahman *et al.* [19] built Machine learning-based prediction model on the top of a set of code specification metrics proposed in compare with Code Smell proposed by Sharma *et al.*, and they claimed that specifications perform better while predicting human-verified defects in Puppet scripts. Lu, Xu, Wu *et al.* [20-22] focused on Docker smell, but mostly towards impact analysis and smell relationship evaluation in terms of correlation and impact on project specifications such as size, business technologies (for example PHP, CSS) and complexity, and they directly applied an open-sourced lint tool called *Hadolint* to detect SC-smell, *i.e.* Shell Script best practice violations, and DL-smell, *i.e.* implementations against officially proposed best practice of Dockerfiles.

Most of the works shed light on general implementation problems, while Rahman *et al.* [17] tried to analyze potential security smells in infrastructure code, providing a brand-new aspect of cloud smell detection other than traditional topics such as readability, maintainability, and testability.

B. RQ2: Impact and Refactoring

Are the impact and refactoring approach of such smells discussed in literatures?

Smell distribution is evaluated in two-third of the IaC papers. In contrast, Co-occurrence as well as the impact of different kinds of cloud and service smells are rarely discussed, which reveals a huge gap in the analysis of the cause and nature of such smells. In the meantime, to the best of the author's knowledge, there lacks empirical studies of the refactoring method and impact of these smells. Additionally, the relationship between normal smells, service reliability and other key aspects and critical metrics such as energy and security are not investigated empirically, making it totally unable to analyze the actual effect of implementing the papers above in real life scenarios.

The quantification works look even less supportive in Microservice smell literatures. As quantification methods are not proposed in numerous literatures, the researchers are not able to determine the actual distribution of such smells, as well as their real impact of the whole service system.

C. RQ3: Future Development

What would the future development of cloud- and service-related smell research be like?

The main topic of cloud- and service-related smells stands on IaC and microservice smell detection, and it is a currently emerging area. There are already several state-of-the-art rule- and metric-based detection approaches for security, implementation, and design in different popular technologies. However, the impact and the actual effect of these works remain unknown, and more works are expected to be done in those fields. The author believe that numerous empirical and impact analysis research would come to us in the near future.

V. CONCLUSION

Cloud and Microservice based applications are gaining attention in engineering as well as academic research, but the bad practices exist in such applications have not been systematically discussed, and their impact to actual systems remains unknown, making it an important task to accomplish.

The author uses Code Smell, *i.e.* the sub-optimal implementation and design choices, as an aspect to look through the bad practice of Cloud and Microservice based applications. This paper investigates 33 infrastructure configuration smells and 34 microservice smells in a form of an SLR, and provides 2 appendixes concerning Cloud and Microservice smells. Additionally, this paper predicts future development of this field according to actual works done and the gaps to be filled.

REFERENCES

[1] F. Palomba, A. Panichella, A. Zaidman, R. Oliveto and A. De Lucia, "The Scent of a Smell: An Extensive Comparison Between Textual and Structural Smells," in *IEEE Transactions on Software Engineering (TSE)*, vol. 44, no. 10, pp. 977-1000, Oct. 2018.

[2] F. Palomba, D. A. A. Tamburri, F. Arcelli Fontana, R. Oliveto, A. Zaidman and A. Serebrenik, "Beyond Technical Aspects: How Do Community Smells Influence the Intensity of Code Smells?", in *IEEE*

Transactions on Software Engineering (Early Access) (TSE).

[3] M. Tufano et al., "When and Why Your Code Starts to Smell Bad (and Whether the Smells Go Away)," in *IEEE Transactions on Software Engineering (TSE)*, vol. 43, no. 11, pp. 1063-1088, Nov. 2017.

[4] F. Palomba, G. Bavota, M. Di Penta, R. Oliveto, A. De Lucia and D. Poshyvanyk, "Detecting bad smells in source code using change history information," 2*013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 268-278.

[5] F. Palomba, M. Zanoni, F. A. Fontana, A. De Lucia, and R. Oliveto, "Smells Like Teen Spirit: Improving Bug Prediction Performance Using the Intensity of Code Smells," *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2016, pp. 244-255.

[6] F. A. Fontana, V. Ferme, M. Zanoni and R. Roveda, "Towards a prioritization of code debt: A code smell Intensity Index," *2015 IEEE 7th International Workshop on Managing Technical Debt*, pp. 16-24.

[7] N. Sae-Lim, S. Hayashi, and M. Saeki, "Context-based code smells prioritization for prefactoring," *2016 IEEE 24th International Conference on Program Comprehension (ICPC)*, pp. 1-10.

[8] F. Mu, L. Shi, W. Zhou, Y. Zhang, and H. Zhao, "NERO: A Text-based Tool for Content Annotation and Detection of Smells in Feature Requests," *2020 IEEE 28th International Requirements Engineering Conference (RE)*. pp. 400-403.

[9] H. Jebnoun, H. Braiek, M. Rahman and F. Khomh, "The Scent of Deep Learning Code: An Empirical Study," *2020 17th International Conference on Mining Software Repositories (MSR)*, Oct. 5-6, Seoul, Republic of Korea. ACM. 11 pages.

[10] A. Borrelli, V. Nardone, G. A. Di Lucca, G. Canfora, and M. Di Penta. "Detecting Video Game-Specific Bad Smells in Unity Projects". *2020 17th International Conference on Mining Software Repositories (MSR)*, Oct. 5-6. ACM. 11 pages.

[11] T. Sharma, M. Fragkoulis and D. Spinellis, "Does Your Configuration Code Smell?", *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*. pp. 189-200.

[12] D. Taibi and V. Lenarduzzi, "On the Definition of Microservice Bad Smells," in *IEEE Software*, vol. 35, no. 3, pp. 56-62, May/June 2018.

[13] A. Tahir, J. Dietrich, S. Counsell, S. Licorish, A. Yamashita, "A large scale study on how developers discuss code smells and anti-pattern in Stack Exchange sites", *Information and Software Technology (IST)*, Vol. 125, 2020, pp.106333.

[14] E. V. d. P. Sobrinho, A. De Lucia and M. d. A. Maia, "A systematic literature review on bad smells — 5 W's: which, when, what, who, where" in *IEEE Transactions on Software Engineering*.

[15] AWS Outage that Broke the Internet Caused by Mistyped Command. Available:https://www.datacenterknowledge.com/archives/2017/03/02/aws-outage-that-broke-the-internet-caused-by-mistyped-command. Accessed: 2020/10/28.

[16] J. Schwarz, A. Steffens, and H. Lichter, "Code Smells in Infrastructure as Code," *2018 11th International Conference on the Quality of Information and Communications Technology*, pp. 220-228.

[17] A. Rahman, C. Parnin and L. Williams, "The Seven Sins: Security Smells in Infrastructure as Code Scripts," *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pp. 164-175.

[18] I. Kumara, Z. Vasileiou, G. Meditskos, D. A. Tamburri, W.-J. Van Den Heuvel, A. Karakostas, S. Vrochidis and I. Kompatsiaris. "Towards Semantic Detection of Smells in Cloud Infrastructure Code, " *2020 10th International Conference on Web Intelligence, Mining and Semantics (WIMS)*. pp. 63–67.

[19] A. Rahman, L. Williams, "Source code properties of defective infrastructure as code scripts", *Information and Software Technology (IST)*, vol. 112, 2019, pp. 148-163.

[20] Y. Wu, Y. Zhang, T. Wang, and H. Wang, "Characterizing the Occurrence of Dockerfile Smells in Open-Source Software: An Empirical Study," in *IEEE Access*, vol. 8, pp. 34127-34139, 2020.

[21] Z. Lu, J. Xu, Y. Wu, T. Wang, and T. Huang, "An Empirical Case Study on the Temporary File Smell in Dockerfiles," in *IEEE Access*, vol. 7, pp. 63650-63659, 2019.

[22] J. Xu, Y. Wu, Z. Lu and T. Wang, "Dockerfile TF Smell Detection Based on Dynamic and Static Analysis Methods," *2019 IEEE 43rd Annual Computer Software and Applications Conference*. pp. 185-190.

[23] A. Rotem-Gal-Oz, *SOA Patterns*. Shelter Island, NY: Manning, 2012.

[24] J. Král and M. Zemlicka, "The Most Important Service-Oriented Antipatterns," *2007 International Conference on Software Engineering Advances (ICSEA)*. 25-31 Aug. IEEE. pp. 29.

[25] J. Garcia, D. Popescu, G. Edwards, and N. Medvidovic, "Toward a Catalogue of Architectural Bad Smells," *2009 International Conference on the Quality of Software Architectures*. Jun. 24-26. pp. 146-162.

[26] J. Král and M. Zemlicka, "Popular SOA Antipatterns," *2009 Computation World: Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns*. Nov. 15-20. IEEE. pp. 271-276.

[27] M. Nayrolles, N. Moha and P. Valtchev, "Improving SOA antipatterns detection in Service Based Systems by mining execution traces," *2013 20th Working Conference on Reverse Engineering (WCRE)*. pp. 321-330.

[28] F. Palma, N. Moha and Y.-G. Guéhéneuc, "UniDoSA: The Unified Specification and Detection of Service Antipatterns," in *IEEE Transactions on Software Engineering*, vol. 45, no. 10, pp. 1024-1053, 2019.

[29] F. Palma and N. Mohay, "A study on the taxonomy of service antipatterns," *2015 IEEE 2nd International Workshop on Patterns Promotion and Anti-patterns Prevention (PPAP)*. Mar. 3. pp. 5-8.

[30] M. Richards, *Microservices Antipatterns and Pitfalls*. Sebastopol, CA: O'Reilly, 2016.

[31] A. Carrasco, B. van Bladel, and S. Demeyer, "Migrating towards microservices: migration and architecture smells," *2018 2nd International Workshop on Refactoring (IWoR@ASE)*. Sept. 4. ACM. pp.1-6.

[32] J. Bogner, T. Boceck, M. Popp, D. Tschechlov, S. Wagner and A. Zimmermann, "Towards a Collaborative Repository for the Documentation of Service-Based Antipatterns and Bad Smells," *2019 IEEE International Conference on Software Architecture (ICSA)*. Mar. 25-26. pp. 95-101.

[33] A. Ouni, R. Gaikovina Kula, M. Kessentini, and K. Inoue, "Web Service Antipatterns Detection Using Genetic Programming," *2015 Genetic and Evolutionary Computation Conference*. Jul. 11-15. ACM. pp. 1351-1359.

[34] B. Dudney, S. Asbury, J. K. Krozak, and K. Wittkopf, *J2EE antipatterns*. John Wiley & Sons, 2003.

[35] S. Jones, "SOA anti-patterns," Available: https: //www.infoq.com/articles/SOA-anti-patterns. Accessed: 2020/10/28.