Detecting anti-patterns in a MSA using distributed tracing

Nils Hullegien
4389069

MSc Thesis — Computer Science — Software Technology
5 July 2022

TUDelft

DETECTING ANTI-PATTERNS IN A MSA USING DISTRIBUTED TRACING

A thesis submitted to the Delft University of Technology in partial fulfillment of the requirements for the degree of

Master of Science

by

Nils Hullegien

5 July 2022

The work in this thesis was made in the:

Software Engineering Research Group
Computer Science - Software Technology
Faculty of Electrical Engineering, Mathematics & Computer Science
Delft University of Technology

ING
ING Bank N.V.

Supervisors:	Prof.dr. Arie van Deursen
	Dr. Luís Cruz
	Dr. Asterios Katsifodimos
	Dr. Pieter Vallen (ING)
	Dr. Jonck van der Kogel (ING)
	Kevin van der Vlist PDEng MSc (ING)
	Tom Liefheid (ING)

ABSTRACT

Microservice architectures (MSA) have become a dominant architectural style choice in the service oriented software industry Alshuqayran et al. [2016]. Because of this, as with any other system, some unoptimized approaches might creep into architectures. These are what we call anti-patterns, they can be considered the opposite of design patterns. Furthermore, a microservice architecture can quickly grow to an immense scale due to the number of services.

In this work, we present Luduan, a tool created within ING that provides engineers with insights into their MSA. Using different graph metrics and tracing data, we determine the likelihood of any service containing certain anti-patterns. We validate this methodology by gathering feedback from subject-matter experts, by ways of surveys and one-on-one sessions where Luduan is used as a support tool. Finally, we ask teams that are responsible for services, to figure out whether their service has an anti-pattern or not. We then use these results to fine tune the computations from metrics to anti-pattern likelihood.

ACKNOWLEDGEMENTS

Throughout my thesis at ING, I have been supported by many great people, who I all thank for their help. First, I want to thank Luís Cruz for his help as daily supervisors, more concretely the help with my thesis writing and guidance throughout, as well as moral support and fun discussions. I also want to thank Arie van Deursen for overseeing the research and providing feedback during my milestone meetings. Lastly, I want to thank Asterios Katsifodimos for being part of my thesis committee and co-reading my thesis.

From ING, I want to thank Pieter Vallen and Jonck van der Kogel for being my supervisors and helping me on all fronts, from academic work to technical questions, as well as providing me with connections throughout ING. I also want to thank Tom Liefheid for supporting me throughout the technical side of the thesis, instructing me about the software as well as the infrastructure within ING. Furthermore, I want to thank Kevin van der Vlist for being a daily supervisor at ING, thank you for all the discussions and brainstorm sessions, as well as feedback throughout my thesis. Finally, I want to thank Yaping Luo for helping me at the start of my thesis and bringing me up to speed within the company, I wish you all the best of luck at your new position within ING. Of course, I also want to thank all my other colleagues for their support and feedback.

Besides these people, none of this would have been possible without the support from my family, girlfriend and other friends that have helped me throughout my thesis and the rest of my studies.

CONTENTS

1 INTRODUCTION

Microservice architectures have become a dominant architectural style choice in the service oriented software industry Alshuqayran et al. [2016]. Like every choice in computer science, using a microservice architecture has upsides and downsides. While it helps to create a highly maintainable and independently deployable system, it also adds a layer of complexity and overhead in the network when set up properly. When this is not the case, anti-patterns are introduced, which cause further unneeded overhead and complexity, not only in the network but also for example in communication between teams or in the code itself.

There are design patterns in place for microservice architectures, which act as guidelines for your own architecture. However, as is the case with smaller segments of code, there are also anti-patterns. These anti-patterns can be seen as pitfalls and should be avoided. For code segments, people have tried to automate the detection of these anti-patterns by looking at code smells Velioğlu and Selçuk [2017]. For microservices, this is an unexplored area.

Detecting issues on a service level within a microservices architecture can however be difficult, especially when the system becomes of significant scale. When looking at code, it can be easier to detect anti-patterns automatically, as tools to help achieve this goal exist. For example, many tools are publicly available for Java code Rutar et al. [2004], which give an indication of code smells. In the case of microservices, we don't have this luxury currently. The main goal of detecting anti-patterns is to improve code and therefore avoid issues and reduce complexity. When working with complex systems in a microservices architecture, these issues arise before they can be avoided.

It can be very useful to know issues on a service level. Over time, a lot of software systems grow in an organic way. Core parts of a system form the backbone of the entire system; if they have an issue, it will most likely affect other important parts of the system as well. These centralized services are important to monitor for this exact reason. Having a system to know and understand these will provide a better insight into the MSA, as well as help engineers and architects to make refactoring choices when services are starting to cause issues. Besides this, when a certain service has caused an incident or issue, using a tool to detect anti-patterns will help to investigate whether this incident was a one-off or whether this service might contain one or more anti-pattern(s), in which case a refactoring step might be required. Providing engineers and architects with a tool like this will allow for a better understanding of the architecture, as well as provide a way to determine what services are the most fragile in the architecture.

One of the main sources of information about the interaction between services, is distributed tracing data Parker et al. [2020]. Distributed tracing adds logging data about requests between services. When we are able to interpret and understand this data, this will allow us to get a better understanding of the weak points in the microservices architecture.

In this thesis, we look at detecting anti-patterns in a microservices architecture using distributed tracing data at ING. Over the last years within ING, a microservice architecture has been created as the backbone of all applications. This architecture contains around 1250 services, which makes manually looking through the distributed tracing data by hand a nigh impossible task to do. However, with such a system, there will be weak points that could be crucial to resolve before they become issues, which then could affect the millions of customers. Therefore, it is crucial to attempt to stay ahead of the curve and detect these code smells and anti-patterns automatically. The research questions we will answer in this paper are the following:

- **RQ 1:** Is it possible to use static architectural metrics to identify anti-patterns in the micro-services architecture?
- **RQ 2:** How relevant or useful is the anti-pattern that is detected to the quality of the service, according to the perception of the maintainers of the code?

To automatically detect anti-patterns, we analyse the distributed tracing data and gather metrics from it, after which we combine relevant metrics to determine the likelihood that a certain service contains an anti-pattern in comparison to the other services in the system. Using a case study with 5 employees at ING, a team survey in which 2 teams participated and a survey in which 10 employees participated, we will validate if this solution correctly detects anti-patterns.

In this paper, we will first look related work (Chapter 2) at the solution proposal (Chapter 3), after which we look at the methodology (Chapter 4) used for the solution. After this, we look at the results of the methodology (Chapter 5). Finally, we have discussion and future work (Chapter 6), which are closed out by the conclusion of the thesis (Chapter 7). In the appendix (Section 5.5), we look at the different anti-patterns and how they would score to be implemented within the ING system.

2 RELATED WORK

In this section, we will look at different approaches taken from related work. We look at the main points taken by that work and see how the approach we take differs.

This work is preceded by the work from Hübener et al. [2022], in which Hübener et al. focus on detecting anti-patterns using distributed tracing. In this thesis, we continue this work by testing the methodology that was used in this paper. We look into the anti-pattern profiles that were set up, as well as the threshold values that were previously determined. Furthermore, we update this work to be usable for every employee within ING, to allow them to experiment and provide feedback.

In Palomba et al. [2015], Palomba provides detection strategies for anti-pattern detection within normal code from the literature. In total, 11 anti-patterns are analysed and detected using different methods, like using specific quantiles within metrics for anti-patterns, using Jaccard distance, making use of abstract syntax trees (AST), using metrics over time and change sets. Even though this paper is not about detecting anti-patterns for services, it is still relevant for this work, as anti-patterns on an architectural level and on class level can be applied similarly and therefore their detection methods as well. In this work, we work with anti-patterns on an architectural level and list different anti-patterns that can be detected using our methodology. However, we do this by scoring anti-patterns on their relevance and impact on the MSA at ING. In Palomba et al. [2015], a small subset of anti-patterns was taken and thoroughly analysed.

In Tighilt et al. [2020], Tighilt et al. conduct a systematic literature review and analysis of open-source microservice-based systems to find 16 microservice anti-patterns. These are described and refactoring solutions are provided. This work also provides insights into developer impact and end user impact. This paper is relevant as the mega service and nano service anti-patterns are implemented in this thesis, but it also provides a basis for the literature review on anti-patterns, which can be found in Section 5.5. This paper builds from this work by detecting some of the anti-patterns automatically, as well as by gathering more anti-patterns that could automatically be detected by the methods used in this thesis.

In Bogner et al. [2019], Bogner et al. collect different service-based anti-patterns from literature and categorize them. In total, 36 different anti-patterns were found, which were grouped in the following way:

- **Architectural anti-patterns**, which impact architecture- or design-related aspects of the system.
- **Application anti-patterns**, which impacts interactions of application components to application-level functionality.
- **Business anti-patterns**, which impact interactions with users, business and data.

Bogner et al. provides a list of different studies, as well as a list of service-based anti-patterns. This paper builds upon this by collecting more anti-patterns, and categorizing them by resilience and changeability. This provides a structured, ordered list of anti-patterns that contain a factor of importance, which can help when selecting what anti-pattern to look further into or to implement for the use of automatic anti-pattern detection.

In Kral and Zemlicka [2007], Kral and Zemlicka use risk management to categorize different anti-patterns. Using three different factors, namely the expected risk, the loss the anti-pattern will cause (which can for example be a project failure) and the probability that this loss will occur. These factors are categorized in values like low, high and very high. The anti-patterns that are categorized based on the risk management factors are "no legacy", "standardization paralysis", "business process forever", "sand pile" and "on-line only". The definitions and references for these anti-patterns can be found in Section 5.5

Using risk management to assess the anti-patterns can be a nice approach to determine their impact. In this thesis, instead of risk management, we look at factors that are important within ING. In ING, an MSA was created in order to deliver customer features in an agile way. Employees work in small groups on one or more microservices, which are then delivered as REST APIs. This allows for smaller scope, faster coding and high availability, as is the aim for an MSA. However, this also created overhead for the teams in the form of remote dependencies and cross-team communications. Looking at high availability, this can be the case for a single service or API, however as with most computer systems, the microservice chain is as strong as its weakest service. These services are often the cause of issues, which is why it is crucial to be able to detect these.

To formalise these factors, we label them as resilience and changeability/maintainability, and use them to categorize anti-patterns. We do this for a number of anti-patterns, the result of which can be found in the appendix (Section 5.5).

In Gamage and Perera [2021], Gamage and Perera look into automatically detecting anti-patterns in MSA applications, where they use tracing data to generate dependency graphs and extract metrics from that. After this, they analysed a sample MSA system that they created themselves, as well as open source systems. This sample MSA system was built to contain the anti-patterns they try to detect, following the metrics that they established themselves for the tool. For their metrics, they chose to use strongly connected components, the clustering coefficient and degree centrality for graph algorithms, as well as path length, in-degree and out-degree per service. Using these metrics, they detect 5 anti-patterns, namely the knot, nano service, service chain, bottleneck service and cyclic dependency anti-pattern. For every anti-pattern, they chose one metric and one graph algorithm to detect it.

The general overview of their work is quite similar to this work, however there are some key differences. One of the major parts is that the environments and systems on which the tool is used, are handcrafted to test parts of the system and trigger for certain anti-patterns. This can be a good way to test if the detection is successful, however using the tool on a different environment will give different results every time and the outcome will be unknown. In this work, we use the system that is already present within ING and attempt to find the services that are the most likely to contain an anti-pattern. This approach is discussed in the solution. Another major part is that there is no threshold in place in the results generated by the tool, instead of this the developers are supposed to determine a threshold based on their data and results. In this work, we look for the most likely values for thresholds within ING based on the input from engineers, since our tool is aimed towards the use within ING. However, since the methods are included in this paper, they will also work as a methodology in case that this tool needs to be used on another system. Within ING, this is already good to have, since we can use the methodology for different stage environments to provide accurate results.

3 SOLUTION

In the solution, we use distributed tracing data Bento et al. [2021] to gather metrics from the microservices. In short, it is possible to determine what services are in the system, what endpoints these services have and how these endpoints between services interact with each other. From this information, we can also determine what services have a connection with each other by using their respective endpoints; if the endpoints have interactions, we know that thereby the services also have interactions. First, we look at the core concepts that our solution is built upon using an example. After this, we will discuss the different metrics that are used to detect anti-patterns, after which we will explain which anti-patterns were chosen for this solution. Finally, we will look at the approach to combine the metrics and anti-patterns to get our solution.

3.1 CORE CONCEPTS

To gather all the data for metrics, we use distributed tracing. This is a technique which relates logging information to different parts of the system Bento et al. [2021]. A trace can be seen as a log entry within distributed tracing, which is built up from multiple spans. One span is a single unit of work within a trace, in our case this can be a call from one service to another, which contains information like the source, destination and information about its parent trace.

From this tracing data, as described by Hübener et al. [2022], we can determine what services are connected to each other. A visualisation of this process can be found in Figure 3.1. We see that *Service A* has an operation called *Operation X* and *Service B* has an operation called *Operation Y*. Besides this, we can also determine that *Operation X* consumes *Operation Y*, which in this scenario gives an indication that *Operation X* directly interacts with *Operation Y*. When we have a configuration like this, we can conclude that, since their operations are interacting, *Service A* and *Service B* are also interacting with each other. We denote this by adding an extra predicate called *CALLS*, which is a directed edge between both services. When we do this for all services in the architecture and use this *CALLS* predicate as edges between the services, we can create a directed graph which contains all the interactions between all services. From this graph, we can determine different metrics, which will be discussed below.

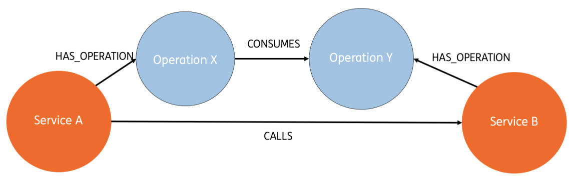


Figure 3.1: Visualisation of the way to connect services from tracing data

3.2 GRAPH METRICS

For the metrics in this solution, we focus on graph metrics. In this section, we describe the metrics we use, namely AIS, ADS, WSIC, SIUC, Betweenness score, LCC and Cycle detection. The first four metrics are computed in code, the last three metrics are created using a library in Java called JGraphT [1]. Since the back-end of Luduan is written in Kotlin, we can make use of all Java libraries, and therefore we can also use JGraphT.

For every service, we want to know about its neighbouring services and its endpoints in the graph. For this, Apolinario de Freitas Apolinário and de França [2020] proposes the metric of *Absolute Importance of a Service* or AIS for short, as well as the metric of *absolute dependence of a service* or ADS for short. The AIS metric counts the number of services that invoke at least one of the endpoints of the service for which the AIS value is computed. The ADS metric counts the number of services that the service in question, depends on. For both of these metrics it holds, that the higher the value, the more connections this service has, the more prominent the service is.

Another metric that is used is proposed by Munialo Munialo et al. [2019], called the "weighted services interface count" or WSIC abbreviated. This metric counts the number of endpoints each service has.

Besides these, a metric proposed by Perepletchikov et al. [2007] called the SIUC, which stands for "service interface usage cohesion", which quantifies cohesion of a given service based on the cohesiveness of the exposed operations. In general, the higher the cohesiveness and therefore the higher the value of the SIUC, the better.

Furthermore, other important metrics need to be taken into account to compare services in the entire system. An example for this is the betweenness score, suggested by Barthelemy Barthélemy [2004]. The metric is computed by taking the shortest path from every service in the system to every other service in the system, and counting how frequently a service is included in all of these shortest paths. Unfortunately, this method is very expensive in terms of computation. In our solution, we have built in a timeout to terminate the betweenness computation if it takes too long. However, for the service metric computation, this timeout will only be reached when there are severe performance issues. When calculating this for the operations however, this timeout is reached more frequently.

Also, we use the local clustering coefficient, or LCC abbreviated. This metric describes the likelihood for a service, that its neighbouring services are also connected to each other. This is calculated for every service in the architecture.

Finally, we look at cycle detection. For every service with a cycle in the architecture, which does not include self cycles, we set this metric value to 1, otherwise we set it to 0.

Using all of these metrics, the system will provide further insights into how services interact and differentiate with nearby services.

3.3 ANTI-PATTERNS

In this solution, we look at a selection of five anti-patterns, namely the mega service, nano service, ambiguous service, bottleneck service and cyclic dependency anti-patterns. This selection of anti-patterns will provide us with insights into the services, as well as give intuitive solutions to resolve any anti-patterns that might arise. The mega service will provide insights into the services in the system that are too large compared to other services. With this, the service can become a single point of failure,

1 https://jgrapht.org/

or a service that is difficult to maintain due to all its complexities and technical debt created by its size. An approach to handle this anti-pattern is to split this big service up into multiple smaller services, each with their own purpose. A service that is flagged with the ambiguous service anti-pattern, usually has a large amount of functionality, but only has a few endpoints for other services to access this. The way to resolve the service affected by this anti-pattern is to make the endpoints less generic, to provide other services with more selective endpoints for the information they require from the ambiguous service. Furthermore, a bottleneck service is a service that is used by too many other services, and therefore becomes a bottleneck in the system and a single point of failure. This anti-pattern can be improved by splitting up the service into smaller services per functionality step or per domain, to allow different services to connect to the segment of the bottleneck service that they are interested in. While this anti-pattern is quite similar to a mega service anti-pattern, this anti-pattern occurs when the service has a high threat of becoming a single point of failure or a bottleneck in the system. This is not necessarily the case for the mega service anti-pattern, as that can also be for nodes in the architecture that are not centralized in the system, but are too large to be considered a microservice.

Finally, cyclic dependencies should not be present in a microservice architecture, as this causes extra confusion and complications in dependencies from one service to another. By checking what services are involved in a cyclic dependency and flagging them, we provide a focus point for the users to look into.

3.4 ANTI-PATTERN DETECTION

When we create this connection between services within the system, we can discover a directed graph of services with their interactions, from which we can then determine metrics.

Each of these metrics is analysed for every anti-pattern that we want to detect, to see whether the results of these metrics would provide useful insights. For example, the mega service anti-pattern could be identified when the values for AIS and ADS are high compared to other services, because that would show that there are many services that have an interaction with the selected service. For the detection of the nano service anti-pattern, this is the opposite, because the values for AIS and ADS need to be relatively low, due to the limited interactions required with other services for a service to be considered a nano service. This procedure is done for all the anti-patterns to determine a so called anti-pattern vector, which shows for each anti-pattern what metrics are relevant and what values services should have for these metrics compared to other services. For this work we use the anti-pattern vector proposed by Hübener et al. [2022], as can be seen in Figure 3.2, however we test the results by reevaluating the factors for every metric in Chapter 4.

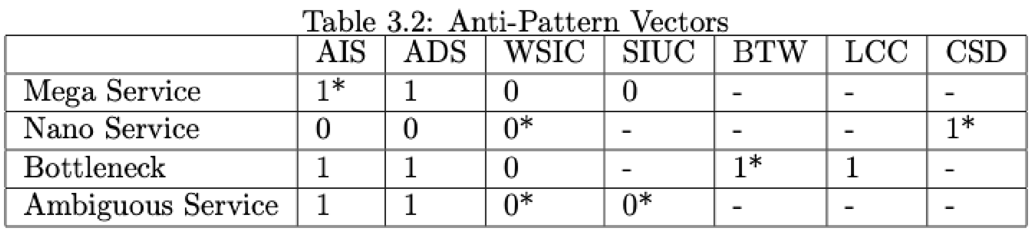
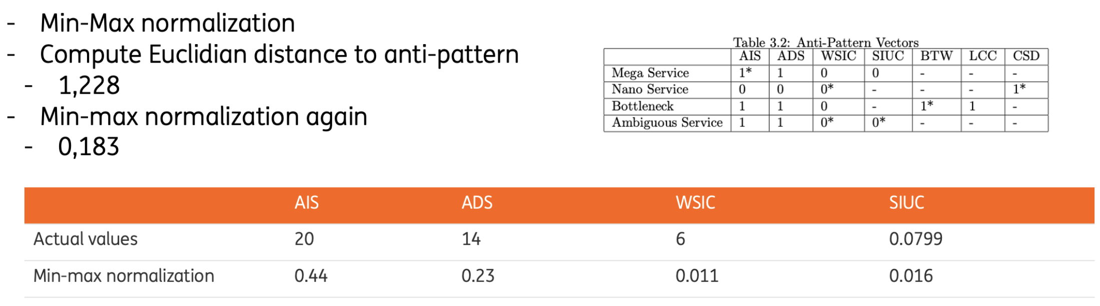


Table 3.2: Anti-Pattern Vectors

	AIS	ADS	WSIC	SIUC	BTW	LCC	CSD
Mega Service	1*	1	0	0	-	-	-
Nano Service	0	0	0*	-	-	-	1*
Bottleneck	1	1	0	-	1*	1	-
Ambiguous Service	1	1	0*	0*	-	-	-

Figure 3.2: The anti-pattern vectors used in this thesis

Using these anti-pattern vectors, we can determine the probability that a service is considered to be affected by a certain anti-pattern. We do this through the process of normalization, which allows us to represent the metrics on a usable scale, to determine what non-normalized values are actually high or low based on all the values in the system. We first normalize the aforementioned metrics for every service, thereby creating a range from 0 to 1 for every metric. After this step, the normalized metrics are used in the anti-pattern vector to calculate the distance from every service to every anti-pattern. Finally, these distances are normalized again to get the relative distance of every service to every anti-pattern relative to others. A short example in practice can be found in Figure 3.3



- Min-Max normalization
- Compute Euclidian distance to anti-pattern
  - 1,228
- Min-max normalization again
  - 0,183

Table 3.2: Anti-Pattern Vectors

	AIS	ADS	WSIC	SIUC	BTW	LCC	CSD
Mega Service	1*	1	0	0	-	-	-
Nano Service	0	0	0*	-	-	-	1*
Bottleneck	1	1	0	-	1*	1	-
Ambiguous Service	1	1	0*	0*	-	-	-

	AIS	ADS	WSIC	SIUC
Actual values	20	14	6	0.0799
Min-max normalization	0.44	0.23	0.011	0.016

Figure 3.3: An example of how the distance to anti-patterns is computed

After this, we can group the normalized distance from each service to every distinct anti-pattern to determine the services that are most likely affected by a certain anti-pattern.

4 METHODOLOGY

In this section, we look at how we have verified the results of the anti-pattern detection in the system. During this work we took three different approaches. In the first section, we look into a survey that was proposed to ING employees to look into their opinion compared to the results Luduan produced. In the second section, we look into one-on-one sessions in Luduan with ING employees, where they think about services they work on or maintain. We look into their opinion on the service, how this relates to other services within the architecture, and look into their opinion and reasoning. In the third and final section, we look into a short questionnaire that was sent to different teams within ING which have services that stand out from others based on their values for the chosen metrics. We asked them to rate their own services based on the anti-patterns that are detected by Luduan and, from this, attempt to determine a weight for each of the metrics in the anti-pattern calculation to get more accurate detection.

The results of these verification processes can be found in the results (Chapter 5) section of the thesis.

4.1 SURVEY

For our survey, we look into different services within the system and identify how employees would rate these services for anti-patterns, based on the metrics within the architecture. We then compare these results with the results generated by Luduan. These results are verified by way of a case study for employees at ING, which includes engineers, architects and team leads. We first explained the different metrics used in the computation of the normalized distance. For this purpose, we used the explanation found in Appendix A as well as the images found in Figure 4.1 and Figure 4.2, after which we provided a subset of services which have been analysed by the application. Before we asked the practitioner about certain services, we explained the anti-pattern in question, to avoid confusion about the definition of an anti-pattern used in the application.

The survey is created in Microsoft Forms and was done anonymously. The results and the survey itself can be found in Appendix A

Introduction to the metrics

When participants start with the survey, they first learned about the different metrics that are used to detect anti-patterns. The following metrics are currently used for the detection: These are the metrics we described in Chapter 3, namely AIS, ADS, WSIC, SIUC, LCC, Betweenness and Cyclic dependency. For this survey, we also included the number of incoming and outgoing tracing messages, as these are metrics that we consider to be useful in the anti-pattern detection as well.

For every metric, a paper is provided for people to read more on the topic Rud et al. [2006] BRANDES and PICH [2007] Omer and Schill [2011] Soffer and Vazquez [2005] Hirzalla et al. [2009] Perepletchikov

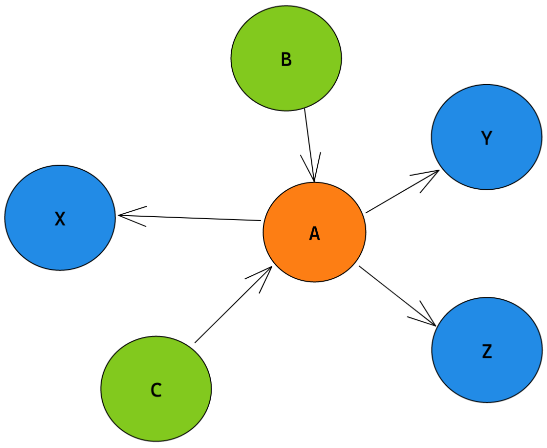


Figure 4.1: AIS/ADS explanation figure

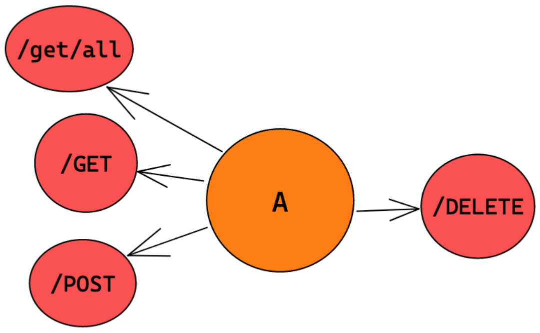


Figure 4.2: WSIC explanation figure

et al. [2007]. For some metrics, a diagram is provided to explain what the metric looks like in the architecture, whereas for others, a textual explanation is provided.

Using the metrics to identify anti-patterns

After reading the explanation of the metrics, the participants will learn about how the metrics are standardized using normalization. Using normalized data is preferred here, as this can give the participant a better understanding of the significance of the values that are created for the different metrics. For example, having a value of 30 will not give much information, as the participant will not know whether this is high compared to other values in the architecture. Providing normalized data remedies this issue.

The metric data is provided normalized per service, since this would give a better insight for the participant without having extra unnecessary metric data. One example of this extra data could be the actual value instead of the normalized value, where another example would be the intermediate values to some calculations. For every anti-pattern, first the anti-pattern itself is explained. After this, a couple of questions are asked about normalized data from services within ING. Using the 5 point Likert scale Joshi et al. [2015] the participant is asked how much certain services should be considered to be anti-patterns, given the data and the definition of that anti-pattern. Each participant will receive 5 services, these are all the same for all participants. Out of these 5 participants, one is considered by the application to be the most likely to be an anti-pattern, one is considered to be as far from the anti-pattern as possible. The other 3 are added based on the normalized data, where one is on 25%, one on 50% and one on 75% of the normalized scale. For every anti-pattern, a new set of 5 different services is chosen following these criteria to give users new cases to consider.

Next, the participant must rate the different metrics based on their importance for the conclusion that they drew. This can again be answered using the 5 point Likert scale of importance. Furthermore, the participant is asked about potential other factors that helped them in their reasoning. Here the user is given an open text field to fill in any other factors, like prior knowledge about a service, or comparing results between services, or anything alike. Finally, the participant can explain whether they have any relation to any of the listed services in their daily work.

4.2 ONE-ON-ONE SESSIONS

During one-on-one sessions, engineers were asked about a service that either they are currently working on within ING, or that they have worked on prior. Therefore, these engineers will have an understanding of the service in question. First, an introduction to the different metrics and anti-pattern computation is given, where the participant can ask questions directly. After this, an expectation is requested from the participant about the service we look at, how this service is expected to score on anti-patterns compared to other services. Then, the participant is asked to look through Luduan and explain their thought process about the application and the results they see. Furthermore, we look together at the results that Luduan generated, compare them to the expectation of the participant and ask how they think the results compare to their expectation. Following this, we ask the participant to look through the code or architecture of the code of the service they chose (if possible). Based on this, we ask the participant whether they would change anything about this, knowing the results Luduan presented. Finally, we ask the participant for general feedback about Luduan, as well as the survey to improve it.

These one-on-one sessions were all done with engineers at ING through Microsoft Teams and took around 30 minutes each.

4.3 SERVICE ANALYSIS BY MAINTAINING TEAMS

In the final questionnaire we did for this paper, we ask different teams within ING to rate their own services for the anti-patterns that are also analysed by Luduan. By doing this, we get an indication what teams consider to be good or bad services. From these results, we analyse the services that were flagged by the team and correlate the metrics from these services with their respective anti-patterns. We choose the teams we email by analysing the metrics within Luduan of all the services within ING. From the 1042 services that are present in the system, we take the 100 services with the highest values for every metric and the 100 services with the lowest values for every metric. Following this, we exclude services that are obvious non-real services, which are identified through their name. For example, services starting with a dummy tag are considered to be testing versions of the real services, and are therefore removed from this top 100. From all of these subsets of services, we look at the services that occur the most frequent in different subsets. This would indicate that these services have values in the extremities of the system quite often, which can be an indication that we are looking at a service with an anti-pattern. From analysis, it turned out that the best result was to take services which occurred at least 4 times in these subsets, which resulted in 162 services in total. From these services, we attempt to find the team that is responsible for this. For some of these services, this information was not available, while there were also teams that were responsible for more than one of the services in the filtered subset. In total, this resulted in 51 teams that we could reach out to.

To these teams, we sent an email asking them to rate all of their services based on the anti-patterns within Luduan. We provided an explanation for the different anti-patterns to inform users about them.

5 RESULTS

In this section, we will look at the results from the different questionnaires we discussed in the methodology section (Chapter 4). We will look to answer our

First we will look at the results of the survey, based on the results in Appendix A. After this, we will look at the insights from the one-on-one sessions. Furthermore, we will discuss the reactions we got from the emails to teams that have a responsibility for services. From these reactions,we see how we can improve our anti-pattern vectors. Following this, we will provide a general conclusion about the results from this thesis. Finally, we will show different anti-patterns that can be used in future work.

5.1 SURVEY RESULTS

In this section, we look at the results from the survey. This survey was taken with only ING employees with different functions within the company, ranging from intern and engineer to chapter lead. More details about this survey can be found in the methodology section (Chapter 4). The questions from the survey can be found in the Appendix A.

In total, 10 employees filled in this survey. The extended results from this survey can be found in Appendix A. First, participants were asked to put in their expectation of how likely a service was considered to contain a certain anti-pattern. When comparing these results to the results from Luduan, we can see that the services for the mega service had the most overlap with the results, whereas the bottleneck and ambiguous service subsets contains the biggest differences. Furthermore, we asked for every anti-pattern, which metrics were considered the most useful by the participant. For both mega-service and nano-service, the importance of the trace metrics stood out the most. Both of these metrics are currently not used in the computation, but gathering from this survey, they should be considered. For the ambiguous service, the metric for the WSIC (number of endpoints for a service) was the only one that was considered to be extremely useful, although AIS and ADS were also considered important. Besides this, no metrics were considered to be helpful in the detection. For the bottleneck service anti-pattern, the betweenness score was considered to be very important, as well as the number of messages. The value for SIUC, which is currently considered to be extra important, was considered to not be important at all from the survey. After each of the anti-patterns, we asked the participants whether they had any relation with any of the mentioned services. Only once, a participant was a direct contributor to one of the services, more often the participants either have heard from the services or have never heard of them.

In general, the variance of the results about the importance of the metrics is quite high. This can be related to different opinions from different employees, but it can also come from a misunderstanding of either the metrics, the anti-pattern or the question itself. In some cases, we can therefore not conclude definitive answers.

5.2 ONE-ON-ONE SESSION RESULTS

In this section, we look at the results of the one on one sessions.

In total, 5 employees took part in these sessions, in which a total of 6 services were discussed. First, participants were asked to discuss how their service would compare to other services within ING in terms of the discussed microservices. All the services were expected to hold up well compared to other services. For all participants, using Luduan turned out to be difficult without any introduction, however after an explanation of the metrics and how Luduan works, everyone was able to navigate the tool to gather the correct information.

When looking at the metrics, it was often difficult for users to put the values of the metrics into perspective. Showing users the range of values that were present within the system cleared this up, however this is something that might need to be improved to help users understand the application better.

When checking the distance to an anti-pattern for the selected service, it turned out that there was quite some noise in the dataset, which caused some values for the distances to either be out of proportion and therefore difficult to estimate for participants. When explaining what happened with the outliers, we reevaluated the result together based on how the data looks without the outliers. In this case, the results were much more favourable, as the outliers caused services to be more likely to be an anti-pattern. When looking at it in this way and reasoning how the service compares to others within the ING architecture, we concluded together that the services were doing well. All the participants stated that the results that Luduan produced were in general in sync with their own belief of the service. Some expected the distance to the mega service and/or the ambiguous service to be lower, as their service was quite small. The reasoning for this was that there were outliers in the data that drove the percentages up, when looking at the full graph with the scores for other services it turned out that these services were still on the low end of the services within ING, which gave the participants a better understanding of the data.

5.3 RESULTS FROM E-MAILS TO RESPONSIBLE TEAMS

In this section, we look at the responses from the e-mail that were directed to teams that are responsible for at least one service with extreme metric values within ING. Furthermore, we determine how we can use these responses to get a better approximation of the weight of metrics in the anti-pattern vectors. In total, 51 teams were asked to answer this question based on their services. Out of these, 2 teams responded with at least one service. We then follow the process that was presented in the methodology section (Chapter 4). From these services, there were 2 services that were registered as mega services by their teams, where both teams stated that they would consider their service to be a mega service and a bottleneck service when compared to the rest of the services within ING. Looking into the results from Luduan, this was quite the opposite, as both services were not identified to contain any of the detected anti-pattern.

5.4 GENERAL CONCLUSION FROM RESULTS

In this section, we look at the results we've gathered and see if we can draw a conclusion from this.

In general, as stated before in the threats to validity, we unfortunately did not get enough responses for all of our experiments to draw any decisive conclusions. We can however claim that the methodology can be used to get a more decisive result when more responses are received.

From the survey, we can state that certain metrics were expected to be not be useful and some are more useful than expected. For example, the number of messages from tracing weren't included before in the computation for anti-patterns. However, from the results we can deduce a suggestion that this metric should be included for all anti-patterns as most participants determined this to be an important metric.

From the one-on-one surveys, we can conclude that the general results from Luduan are quite similar to the expectation of participants. In some cases, the participants are harsher on their service than Luduan states, especially when assessing whether a service is a mega service. We did however have some issues regarding outliers in the data, meaning that the results from Luduan were higher than expected. After explaining this during the sessions and explaining how the service compares to other services, the participants got a better understanding of the situation and the results lined up with their view on the service.

From the team email results, the number of respondents is the biggest problem. The results we gathered are quite interesting, as both participants expected their survey to be a mega service and a bottleneck service, while Luduan did not think this. This could be an indication that other metrics need to be taken into account, the weight of certain metrics need to be increased, or potentially even a misunderstanding of the anti-patterns and the relation of their service compared to other services within the architecture.

5.5 DIFFERENT ANTI-PATTERNS IN MICRO-SERVICE ARCHITECTURES

In this section, we look at different anti-patterns and look at how relevant they are based on important factors in the infrastructure.

For each anti-pattern in this table, we look into two factors, namely resilience and changeability. In the table, these factors are shortened to *RES* and *CHANGE*. These factors are listed on a scale from 1 to 5 for how much the specified anti-pattern affects these factors. When an anti-pattern is given the value of 1, this means that the anti-pattern affects this factor almost not at all. When it is 5 this means that for the specified factor, it is crucial that the anti-pattern is resolved.

Anti-pattern name	**Aliases**	**Sources**	**RES**	**CHANGE**
Ambiguous Interface		Garcia et al. [2009], de Andrade et al. [2014]	1	2
Ambiguous Service	Ambiguous Web Service, Ambiguous Name	Palma et al. [2014], Ouni et al. [2015], Ouni et al. [2017], Palma and Mohay [2015]	2	2
API Versioning		Taibi and Lenarduzzi [2018]	3	5
Big Bang	Vendor lock-in	Král and Zemlicka [2009]	2	2
Bloated Service		Palma and Mohay [2015]	3	2
Bottleneck Service		Nayrolles et al. [2013], Palma and Mohay [2015]	4	3
Business Process Forever	No Businessmen Involvement	Král and Zemlicka [2007]	2	4
Chatty Service	Chatty Web Service	Palma et al. [2014], Ouni et al. [2015], Ouni et al. [2017], Nayrolles et al. [2013], Palma and Mohay [2015]	4	2
Client Completes Service	Incomplete Service	Dudney et al. [2003]	3	3
Connector Envy		Garcia et al. [2009], de Andrade et al. [2014]	4	5
CRUDy Interface	Maybe It is Not RPC	Ouni et al. [2017], Palma and Mohay [2015]	2	3
Cyclic Dependency		Taibi and Lenarduzzi [2018], Tighilt et al. [2020]	5	5
Data-Driven Migration		Richards [2016]	1	2
Data Service	Data Web Service	Palma et al. [2014], Ouni et al. [2017], Ouni et al. [2015], Palma and Mohay [2015]	2	4
Duplicated Service	Duplicated Web Service, Too many Cooks in the SOA	Palma et al. [2014], Palma and Mohay [2015], Jones [2006]	2	5

Anti-pattern name	Aliases	Sources	RES	CHANGE
Extraneous Adjacent Connector	Redundant PortTypes	Garcia et al. [2009], de Andrade et al. [2014], Palma and Mohay [2015]	2	3
ESB (enterprise service bus) usage		Taibi and Lenarduzzi [2018]	2	2
Golden Hammer	When in doubt, make it a service	Dudney et al. [2003]	1	2
Hard-Coded Endpoints	Point to Point Web Services	Taibi and Lenarduzzi [2018], Jones [2006], Tighilt et al. [2020]	3	5
Inappropriate Service Intimacy		Taibi and Lenarduzzi [2018]	2	3
Low Cohesive Operations		Palma and Mohay [2015]	3	3
Megaservice	Multiservice, The God Object, God Object Web Service, Big Ball of Mud	Taibi and Lenarduzzi [2018], Dudney et al. [2003], Ouni et al. [2017], Palma et al. [2014], Ouni et al. [2015], Nayrolles et al. [2013], Palma and Mohay [2015], Tighilt et al. [2020]	5	5
A Million Services all in a row		Král and Zemlicka [2007]	3	4
Nanoservice	Tiny Service, Refactor Mercilessly, Fine Grained Web Service, Fine Grained Service, Fine-Grained Services, Fine-Grained Interfaces	Rotem-Gal-Oz [2012], Dudney et al. [2003], Palma et al. [2014], Král and Zemlicka [2009], Ouni et al. [2015], Ouni et al. [2017], Nayrolles et al. [2013], Palma and Mohay [2015], Tighilt et al. [2020]	3	4

Anti-pattern name	**Aliases**	**Sources**	**RES**	**CHANGE**
Legacy	Nothing New, Same Old Way	Král and Zemlicka [2007], Král and Zemlicka [2009], Rotem-Gal-Oz [2012]	4	5
On-Line Only	No batch processing	Král and Zemlicka [2007]	2	2
Overstandardized SOA	Standardization Paralysis, Vendor-Lock-In	Král and Zemlicka [2009], Král and Zemlicka [2007]	2	3
Sand Pile		Král and Zemlicka [2007], Palma and Mohay [2015]	5	4
Scattered Parasitic Functionality		Garcia et al. [2009], de Andrade et al. [2014]	4	5
Service Chain	Percolating Process	Nayrolles et al. [2013], Jones [2006]	3	4
Shared Libraries		Taibi and Lenarduzzi [2018], Tighilt et al. [2020]	3	4
Shared Persistency		Taibi and Lenarduzzi [2018]	3	3
The Shiny Nickel		Jones [2006]	2	2
Timeouts	Dogpiles	Tighilt et al. [2020], Richards [2016]	5	4
Silver Bullet	Web Services Will Fix Our Problems	Dudney et al. [2003]	1	4
SOAPY Business Logic	Muddy Tiers	Dudney et al. [2003]	1	3
Stovepipe Service	Multilayer Service	Dudney et al. [2003], Palma and Mohay [2015]	3	3
The Knot		Rotem-Gal-Oz [2012], Nayrolles et al. [2013], Palma and Mohay [2015]	3	4
Transactional Integration		Rotem-Gal-Oz [2012]	4	3
Wrong Cuts	Layered msa	Taibi and Lenarduzzi [2018], Tighilt et al. [2020]	2	2

6 | DISCUSSION AND FUTURE WORK

In the first section, we answer our research questions. Furthermore, in the second section, we look at challenges in this work and find improvements that could be made. In the final section, we will look into how the tool was perceived by employees from ING.

6.1 IMPLICATION OF THE THESIS

During the thesis, many steps were taken to improve the visibility of the tool within ING. At first, the tool was a proof of concept, running on a snapshot of data on a local computer. Currently, we have a working tool running on the latest software used within ING, operating on three different testing environments for engineers to use, using live data to provide near real time insights for all services which provide tracing data.

To establish the tool and to gather feedback from potential users, different presentations and demos were given during the time of this thesis. People involved in these presentations range from engineers and software architects of all levels of experience to team leads and employees from many teams within ING. This involves national, but also international employees. Personally, presenting to these employees with different perspectives on the use cases for the tool has given me a better understanding of how important an audience can be to get your presentation message across. In every presentation, the feedback has been overwhelmingly positive and the suggestions to the tool were helpful. From this, we conclude that the tool can be useful for any type of employee that is working with an MSA. For an engineer, it is informing to see the status of your own service, while for an architect it can be a backbone in deciding what part of the architecture needs to be refactored and with what reasoning.

When thinking outside of ING, the results show that this will help every company that has an MSA implementation with distributed tracing. Currently, the thresholds for when a service is considered an antipattern are especially determined for within ING, however the same methodology that is used in this thesis, can be applied to any MSA.

6.2 ANSWERING THE RESEARCH QUESTIONS

Our first research question, as described in Chapter 1, states the following: Is it possible to use static architectural metrics to identify anti-patterns in the microservices architecture? A short answer to this question is yes, we are able to use architectural metrics to identify anti-patterns within the MSA of ING. Using the methodology described in Chapter 4, we are able to provide users with new insights into the weaknesses of their services, as can be found in Chapter 5.

Our second research questions, as described in Chapter 1, states the following: How relevant or useful is the anti-pattern that is detected to the quality of the service, according to the perception of the maintainers of the code? To answer this question, we look into the different results that we have gathered. From the one-on-one expert consultation, we gathered that users found the tool to be interesting and useful, as well as an improvement to the current set of tools that is available to detect issues within ING. The expectations from the experts were similar to the results that were presented by the tool.

6.3 IMPROVEMENTS

In this section, we discuss improvements for the thesis, the factors that could be hindering the results, future work and the implications of the work. First, the biggest problem point is the number of participants in each of the surveys that were taken. To get a better and more accurate outcome, getting more employees involved would really benefit the outcome and create a better result. The expected reason behind this is the complication of the survey and the topic, as prior knowledge was required to fill in the survey. This meant that this information needed to be presented concisely, which we succeeded in, but even then the survey was difficult to do. To improve this, we decided to implement the one-on-one sessions, because this way we are able to explain the topics to participants in a better way while also allowing them to ask questions throughout. This meant that participants were able to get a better understanding. Another potential improvement is to schedule a bigger meeting at an event, provide a similar explanation in a classical manner, and then ask participants to fill in the survey. This way, participants can ask questions whenever they misunderstand something, but are also pushed to fill and complete in the survey. By using the different methods to gather information, we are able to get a better understanding of the directions to implement to improve Luduan. To draw a clear conclusion, it will however be required to run similar experiments with a larger number of participants.

Secondly, in order to acquire new insights for the anti-pattern detection and to give more tailored results to address them, more metrics need to be included in the tool. This will provide new insights and will help with critical thinking about anti-pattern detection at ING. Besides, it will draw the attention of other groups within ING, who will provide new insights to consider. It would also allow us to choose different anti-patterns to detect and provide a good anti-pattern vector to reliably detect the anti-pattern. An example of this is the use of trace metrics, using the number of messages turned out to be a metric that was already used in Luduan but not implemented in the anti-pattern vectors. These metrics allow us to improve and add to the current anti-pattern detection.

Thirdly, using more anti-patterns would help users understand how to improve a service. For example, introducing clique detection adds new information about the surrounding nodes, compared to strongly connected components and clustering coefficient. A number of additions have been proposed in Section 5.5.

Fourthly, instead of using direct service-to-service connection info, other data such as database or event stream connection info would achieve the previously discussed points. For example, using datasets that show the connections the service has outside other services, like databases or Apache Kafka, would help to determine whether the service is following other principles in the micro-service architecture.

Fifthly, connecting Luduan with data from incidents and change sets within ING could provide great insights into how services are behaving. This would also allow employees to see whether an incident is a one-off for that service, or whether that service should get a bigger refactoring to resolve anti-patterns.

Finally, the results of services were skewed in a handful of cases, because the entire architecture was taken into account when normalizing the metrics and determining the distance to anti-patterns. Introducing a method to cluster services in a way such that the clustered services are responsible for the same task, will help to identify issues and restrictions in certain tasks performed by clients.

7 CONCLUSION

In this thesis, we looked at the detection of anti-patterns in an MSA using distributed tracing data at ING. We use different graph metrics on these services and detect anti-patterns from them. We looked at the mega service, nano service, bottleneck service and ambiguous service anti-pattern within this architecture.

Furthermore, by taking three different approaches, we tested the chosen methodology. We were able to derive that certain metrics are considered more important than others for certain anti-patterns. Besides this, we were able to use the opinion and perception of individual engineers and teams to determine what services are considered anti-patterns. From this, we were able to provide better detection for anti-patterns within Luduan and therefore improve the awareness of anti-patterns.

We also looked at different anti-patterns that could potentially be detected, either using the current dataset or by including different metrics. Implementing these anti-patterns will allow Luduan to be useful for years to come within ING. Also, by including new datasets, the integration of Luduan with already existing tools at ING that have different purposes, will increase the visibility of the tool as well. During this thesis, I've made Luduan available for everyone within ING, while also exposing the tool to lots of potential users at internal talks, conferences and demos. The consensus during these talks was that people like the tool, see the potential it has and are keen to use it, while also supporting us to spread the word even further within ING.

BIBLIOGRAPHY

Alshuqayran, N., Ali, N., and Evans, R. (2016). A systematic mapping study in microservice architecture. In *2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA)*, pages 44–51.

Barthélemy, M. (2004). Betweenness centrality in large complex networks. *The European Physical Journal B*, 38(2):163–168.

Bento, A., Correia, J., Filipe, R., Araujo, F., and Cardoso, J. (2021). Automated analysis of distributed tracing: Challenges and research directions. *Journal of Grid Computing*, 19(1):9.

Bogner, J., Boceck, T., Popp, M., Tschechlov, D., Wagner, S., and Zimmermann, A. (2019). Towards a collaborative repository for the documentation of service-based antipatterns and bad smells. *2019 IEEE International Conference on Software Architecture Companion (ICSA-C)*, pages 95–101.

BRANDES, U. and PICH, C. (2007). Centrality estimation in large networks. *International Journal of Bifurcation and Chaos*, 17(07):2303–2318.

de Andrade, H. S., Almeida, E., and Crnkovic, I. (2014). Architectural bad smells in software product lines: An exploratory study. In *Proceedings of the WICSA 2014 Companion Volume*, pages 1–6.

de Freitas Apolinário, D. R. and de França, B. B. N. (2020). Towards a method for monitoring the coupling evolution of microservice-based architectures. In *Proceedings of the 14th Brazilian Symposium on Software Components, Architectures, and Reuse*, SBCARS ’20, page 71–80, New York, NY, USA. Association for Computing Machinery.

Dudney, B., Asbury, S., Wittkopf, K., and Krozak, J. K. (2003). *J2EE antipatterns*. John Wiley & Sons.

Gamage, I. U. P. and Perera, I. (2021). Using dependency graph and graph theory concepts to identify anti-patterns in a microservices system: A tool-based approach. In *2021 Moratuwa Engineering Research Conference (MERCon)*, pages 699–704.

Garcia, J., Popescu, D., Edwards, G., and Medvidovic, N. (2009). Toward a catalogue of architectural bad smells. In *International conference on the quality of software architectures*, pages 146–162. Springer.

Hirzalla, M., Cleland-Huang, J., and Arsanjani, A. (2009). A metrics suite for evaluating flexibility and complexity in service oriented architectures. In Feuerlicht, G. and Lamersdorf, W., editors, *Service-Oriented Computing – ICSOC 2008 Workshops*, pages 41–52, Berlin, Heidelberg. Springer Berlin Heidelberg.

Hübener, T., Chaudron, M. R. V., Luo, Y., Vallen, P., van der Kogel, J., and Liefheid, T. (2022). Automatic anti-pattern detection in microservice architectures based on distributed tracing. In *2022 IEEE/ACM 44th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 75–76.

Jones, S. (2006). Soa anti-patterns.

Joshi, A., Kale, S., Chandel, S., and Pal, D. K. (2015). Likert scale: Explored and explained. *British journal of applied science & technology*, 7(4):396.

Kral, J. and Zemlicka, M. (2007). The most important service-oriented antipatterns. In *International Conference on Software Engineering Advances (ICSEA 2007)*, pages 29–29. IEEE.

Král, J. and Zemlicka, M. (2007). The most important service-oriented antipatterns. pages 29 – 29.

Král, J. and Zemlicka, M. (2009). Popular soa antipatterns. In *2009 Computation World: Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns*, pages 271–276.

Munialo, S., Muketha, G., and Omieno, K. (2019). Size metrics for service-oriented architecture. *International Journal of Software Engineering and Applications*, 10:67–83.

Nayrolles, M., Moha, N., and Valtchev, P. (2013). Improving soa antipatterns detection in service based systems by mining execution traces. In *2013 20th Working Conference on Reverse Engineering (WCRE)*, pages 321–330.

Omer, A. M. and Schill, A. (2011). Automatic management of cyclic dependency among web services. In *Proceedings of the 2011 14th IEEE International Conference on Computational Science and Engineering*, CSE '11, pages 44–51, USA. IEEE Computer Society.

Ouni, A., Gaikovina Kula, R., Kessentini, M., and Inoue, K. (2015). Web service antipatterns detection using genetic programming. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pages 1351–1358.

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

Palma, F., Moha, N., Tremblay, G., and Guéhéneuc, Y.-G. (2014). Specification and detection of soa antipatterns in web services. In *European Conference on Software Architecture*, pages 58–73. Springer.

Palma, F. and Mohay, N. (2015). A study on the taxonomy of service antipatterns. In *2015 IEEE 2nd International Workshop on Patterns Promotion and Anti-patterns Prevention (PPAP)*, pages 5–8.

Palomba, F., Lucia, A., Bavota, G., and Oliveto, R. (2015). Anti-pattern detection: Methods, challenges, and open issues. *Advances in Computers*, 95:201–238.

Parker, A., Spoonhower, D., Mace, J., Sigelman, B., and Isaacs, R. (2020). *Distributed tracing in practice: Instrumenting, analyzing, and debugging microservices.* O'Reilly Media.

Perepletchikov, M., Ryan, C., and Frampton, K. (2007). Cohesion metrics for predicting maintainability of service-oriented software. *Seventh International Conference on Quality Software (QSIC 2007)*, pages 328–335.

Richards, M. (2016). *Microservices antipatterns and pitfalls.* O'Reilly Media, Incorporated.

Rotem-Gal-Oz, A. (2012). *SOA patterns.* Simon and Schuster.

Rud, D., Schmietendorf, A., and Dumke, R. R. (2006). R.: Product metrics for service-oriented infrastructures. In *Hasso-Plattner-Institut, Shaker Verlag*, pages 161–174.

Rutar, N., Almazan, C., and Foster, J. (2004). A comparison of bug finding tools for java. In *15th International Symposium on Software Reliability Engineering*, pages 245–256.

Soffer, S. and Vazquez, A. (2005). Network clustering coefficient without degree-correlation biases. *Physical review. E, Statistical, nonlinear, and soft matter physics*, 71:057101.

Taibi, D. and Lenarduzzi, V. (2018). On the definition of microservice bad smells. *IEEE Software*, 35(3):56–62.

Tighilt, R., Abdellatif, M., Moha, N., Mili, H., Boussaidi, G. E., Privat, J., and Guéhéneuc, Y.-G. (2020). On the study of microservices antipatterns: A catalog proposal. In *Proceedings of the European Conference on Pattern Languages of Programs 2020*, EuroPLoP '20, New York, NY, USA. Association for Computing Machinery.

Velioğlu, S. and Selçuk, Y. E. (2017). An automated code smell and anti-pattern detection approach. In *2017 IEEE 15th International Conference on Software Engineering Research, Management and Applications (SERA)*, pages 271–275.

A SURVEY

In this appendix, the questions and raw results from the survey are included. The images in the explanation are added in Figure 4.1 and Figure 4.2, as these are not clear from the full survey. The names of services have been obfuscated as these services are used within ING.

Table A.1: Mega Services

Service 1	Service 2	Service 3	Service 4	Service 5
Strongly disagree	Neutral	Agree	Disagree	Neutral
Strongly Agree	Agree	Strongly Agree	Neutral	Strongly disagree
Strongly Agree	Disagree	Disagree	Strongly disagree	Strongly disagree
Agree	Disagree	Neutral	Disagree	Disagree
Disagree	Neutral	Disagree	Neutral	Agree
Neutral	Disagree	Agree	Neutral	Disagree
Strongly Agree	Disagree	Neutral	Strongly disagree	Strongly disagree
Agree	Disagree	Agree	Disagree	Disagree
Strongly Agree	Strongly disagree	Disagree	Disagree	Strongly disagree
Strongly Agree	Strongly disagree	Agree	Strongly disagree	Strongly disagree

Table A.2: Mega service metric importance part 1

AIS	ADS	WSIC	SIUC	Betweenness score
Slightly important	Slightly important	Moderately important	Moderately important	Moderately important
Important	Slightly important	Moderately important	Not important	Important
Important	Important	Very important	Not important	Important
Important	Important	Not important	Not important	Not important
Important	Important	Moderately important	Moderately important	Not important
Important	Important	Important	Not important	Moderately important
Important	Important	Important	Not important	Not important
Important	Moderately important	Slightly important	Not important	Moderately important
Important	Important	Important	Important	Moderately important
Very important	Not important	Important	Not important	Important

Table A.3: Mega service metric importance part 2

LCC	Cyclic dependency	Number of incoming trace messages	Number of outgoing trace messages
Not important	Important	Important	Important
Very important	Important	Moderately important	Moderately important
Not important	Very important	Important	Important
Not important	Not important	Important	Important
Moderately important	Not important	Important	Important
Moderately important	Slightly important	Not important	Not important
Not important	Not important	Important	Important
Slightly important	Not important	Important	Moderately important
Moderately important	Moderately important	Important	Important
Not important	Not important	Very important	Very important

Table A.4: Mega service miscellaneous questions

Other factors to determine mega service?	Do you have a relation to any of these services?
Yeah, I have an opinion nj the lcc	No, nothing at all;
incoming messages are close to outgoing messages	No, nothing at all;
	I've heard of them;Yes, consumer of one of the services;
No	I've heard of them;
	I've heard of them;
	No, nothing at all;
-	I've heard of them;
	I've heard of them;
No	I've heard of them;
	No, nothing at all;

Table A.5: Nano Services

Service 1	Service 2	Service 3	Service 4	Service 5
Strongly Agree	Agree	Agree	Neutral	Neutral
Strongly Agree	Neutral	Neutral	Agree	Strongly disagree
Agree	Neutral	Neutral	Neutral	Strongly Agree
Agree	Disagree	Agree	Disagree	Disagree
Agree	Disagree	Neutral	Disagree	Disagree
Agree	Neutral	Disagree	Disagree	Disagree
Agree	Disagree	Agree	Disagree	Disagree
Agree	Disagree	Neutral	Disagree	Disagree
Strongly Agree	Agree	Neutral	Disagree	Strongly disagree
Strongly Agree	Strongly disagree	Strongly Agree	Strongly disagree	Strongly disagree

Table A.6: Nano service metric importance part 1

AIS	ADS	WSIC	SIUC	Betweenness score
Moderately important	Moderately important	Moderately important	Moderately important	Moderately important
Slightly important	Slightly important	Moderately important	Slightly important	Important
Not important	Not important	Important	Very important	Not important
Important	Important	Not important	Not important	Not important
Important	Important	Moderately important	Moderately important	Not important
Important	Important	Important	Important	Moderately important
Moderately important	Moderately important	Slightly important	Not important	Not important
Moderately important	Important	Slightly important	Slightly important	Not important
Very important	Very important	Moderately important	Moderately important	Not important
Important	Not important	Slightly important	Not important	Not important

Table A.7: Nano service metric importance part 2

LCC	Cyclic dependency	Number of incoming trace messages	Number of outgoing trace messages
Moderately important	Moderately important	Important	Very important
Important	Slightly important	Moderately important	Moderately important
Not important	Not important	Not important	Not important
Not important	Not important	Important	Important
Not important	Moderately important	Important	Important
Moderately important	Not important	Not important	Not important
Not important	Not important	Important	Important
Important	Not important	Important	Important
Not important	Not important	Important	Important
Not important	Not important	Very important	Very important

Table A.8: Nano service miscellaneous questions

Other factors to determine nano service?	Do you have a relation to any of these services?
	No, nothing at all;
no	No, nothing at all;
	No, nothing at all;
No	I've heard of them;
	No, nothing at all;
	No, nothing at all;
-	I've heard of them;
	I've heard of them;
no	Yes, consumer of one of the services;I've heard of them;
	No, nothing at all;

Table A.9: Ambiguous Services

Service 1	Service 2	Service 3	Service 4	Service 5
Neutral	Neutral	Neutral	Neutral	Neutral
Agree	Strongly Agree	Agree	Neutral	Agree
Neutral	Agree	Strongly disagree	Neutral	Neutral
Disagree	Agree	Agree	Disagree	Agree
Disagree	Disagree	Neutral	Disagree	Agree
Disagree	Agree	Disagree	Disagree	Agree
Disagree	Neutral	Neutral	Disagree	Agree
Neutral	Disagree	Disagree	Agree	Disagree
Neutral	Disagree	Neutral	Agree	Strongly Agree
Strongly disagree	Strongly disagree	Agree	Strongly disagree	Strongly disagree

Table A.10: Ambiguous service metric importance part 1

AIS	ADS	WSIC	SIUC	Betweenness score
Not important	Not important	Not important	Not important	Not important
Important	Slightly important	Very important	Not important	Important
Not important	Not important	Not important	Not important	Not important
Not important	Not important	Important	Not important	Not important
Important	Important	Moderately important	Moderately important	Not important
Important	Important	Moderately important	Not important	Slightly important
Not important	Not important	Important	Not important	Not important
Moderately important	Not important	Important	Not important	Slightly important
Not important	Not important	Slightly important	Slightly important	Very important
Important	Important	Important	Important	Important

Table A.11: Ambiguous service metric importance part 2

LCC	Cyclic dependency	Number of incoming trace messages	Number of outgoing trace messages
Not important	Not important	Not important	Not important
Important	Not important	Moderately important	Not important
Not important	Not important	Not important	Not important
Not important	Not important	Not important	Not important
Not important	Moderately important	Important	Important
Important	Important	Not important	Not important
Not important	Not important	Not important	Not important
Not important	Not important	Important	Important
Very important	Very important	Slightly important	Slightly important
Not important	Important	Important	Important

Table A.12: Ambiguous service miscellaneous questions

Other factors to determine ambiguous service?	Do you have a relation to any of these services?
	No, nothing at all;
no	No, nothing at all;
I rely more on the name of the API then the metrics	Yes, consumer of one of the services;Yes, direct contributer;
No	I've heard of them;
	No, nothing at all;
	No, nothing at all;
-	I've heard of them;
	I've heard of them;
	Yes, direct contributer;Yes, consumer of one of the services; I've heard of them;
For Service 4, it has cyclic dependency; however, its incoming messages is low. This seems to be a sign of ambiguous service.	No, nothing at all;

Table A.13: Bottleneck services

Service 1	Service 2	Service 3	Service 4	Service 5
Neutral	Neutral	Neutral	Neutral	Neutral
Strongly disagree	Strongly disagree	Strongly Agree	Agree	Neutral
Neutral	Agree	Agree	Strongly disagree	Strongly Agree
Disagree	Disagree	Agree	Disagree	Agree
Disagree	Disagree	Disagree	Disagree	Agree
Disagree	Disagree	Agree	Agree	Disagree
Strongly disagree	Disagree	Strongly Agree	Strongly disagree	Agree
Disagree	Disagree	Agree	Disagree	Agree
Strongly disagree	Strongly Agree	Agree	Disagree	Neutral
Strongly disagree	Disagree	Strongly Agree	Strongly disagree	Agree

Table A.14: Bottleneck service metric importance part 1

AIS	ADS	WSIC	SIUC	Betweenness score
Not important	Not important	Not important	Not important	Not important
Not important	Not important	Not important	Not important	Very important
Slightly important	Slightly important	Important	Not important	Important
Not important	Important	Not important	Not important	Not important
Important	Important	Not important	Not important	Not important
Moderately important	Moderately important	Moderately important	Not important	Important
Not important	Not important	Not important	Not important	Important
Important	Moderately important	Slightly important	Slightly important	Important
Not important	Very important	Moderately important	Moderately important	Not important
Important	Important	Not important	Not important	Important

Table A.15: Bottleneck service metric importance part 2

LCC4	Cyclic dependency4	Number of incoming trace messages4	Number of outgoing trace messages4
Not important	Not important	Not important	Not important
Very important	Moderately important	Not important	Not important
Not important	Not important	Very important	Moderately important
Not important	Important	Not important	Not important
Not important	Not important	Important	Important
Important	Slightly important	Slightly important	Slightly important
Not important	Not important	Important	Important
Slightly important	Slightly important	Moderately important	Moderately important
Not important	Slightly important	Very important	Important
Not important	Not important	Important	Important

Table A.16: Bottleneck service miscellaneous questions

Other factors to determine nano service?	Do you have a relation to any of these services?
	No, nothing at all;
no	No, nothing at all;
	Yes, consumer of one of the services;
No	I've heard of them;
	No, nothing at all;
	No, nothing at all;
-	I've heard of them;
	I've heard of them;
no	No, nothing at all;
	No, nothing at all;

Table A.17: Generic questions

What is your role/profession within ING?	Do you have any questions/feedback?
Architect	
Software developer	The survey assumes in-depth knowledge on the (advanced) metrics which takes some learning to understand them. I'm not sure whether the results of the survey would measure experts knowledge on the services with regards of the metrics or mis-interpreting the metrics
IT Chapter Leadd	
Enterprise Architect for the Infrastructure Domain	not interested in the gift card, in case I win you can donate the value to the ING/Unicef action for Ukranian refugees
Chapter Lead	
Software Engineer	
Intern	
Engineer	
Backend Engineer	Nice Survey with good collection of important APIs. May be try to add more details explanation of what each metrics is and show examples of how a higher or lower number impacts that metric
intern	Maybe it would be good to add picture along the metrics. I had a hard time to visualize the definition of metrics :)

Anti-pattern survey

The survey will take approximately 15 minutes to complete.

* Required

Hello! Thanks for filling in the survey

In total, this survey should take around 15 minutes and is completely anonymous (you will need to fill in your e-mail if you want to have a chance to win a gift card, however this is optional).

My name is Nils Hullegien, I am a master student from the TU Delft. My thesis at is about **detecting anti-patterns in our service architecture using distributed tracing data**. The application, called Luduan (a collaboration between team Cerebro and Architects as well), is already running on the architecture, please feel free to have a look, any feedback will be appreciated ( )! The application is meant to be a tool to give users hints at services containing anti-patterns. I believe this tool can be very useful for every employee that has any interaction with services.

In this survey, I want to **test the methods that are used for the anti-pattern detection**. Since the results the application computes do not have a "ground truth" (meaning that we can never say for certain that the methods used are 100% correct in all examples), it is essential to test out the methods used in the computations. This is one of the main reasons why this survey is very important for Luduan within , as well as for my thesis.

In short, this survey will first contain some background information about the metrics that I've used, after which these metrics will be applied to four different anti-patterns. For each of these, an introduction will be given as well, after which 5 services from will be presented with their metrics. For every one of these, try to estimate up to what degree a certain service you would consider to be an anti-pattern or not. Remember that there are no wrong answers.
Finally, for every anti-pattern, there will be a couple of questions about the importance of the different metrics that defined your decision.

All the metrics are computed per service, meaning that every service will have **their own separate set of metrics**.

For the metric explanations, I will make use of visualisations of a graph structure that represents the services at . In this graph, **the services are represented as the nodes and the connections/edges are represented by API calls** from one service to another. The visualisations can be found at the bottom of the two metrics explanation pages.

Remember that you have the chance to win a 30 euro gift card or a 20 euro gift card by filling in this survey and leaving your e-mail at the end of the survey!

Metrics explanation (part 1 of 2)

In this section we look at the first 5 metrics that were used in the detection of anti-patterns.

Looking at the visual representation of a part of the service graph in the image, we can see 6 different services with directed edges towards each other (for example, A can go to Y but not vice versa). From service A, we can see that there are 3 services that are going out of A, marked in blue and with the letters X, Y and Z. This means that the API of A is communicating with the APIs of the blue services. There are also 2 services with connections going into A, marked in green and with the letters B and C.

We use these traits as metrics, looking from the perspective of service A, we call all the green services the **absolute importance of the service** or **AIS** in short. The blue services are called the **absolute dependence of the service** or **ADS** in short. In this example, for service A, the value for AIS is 3 (since there are 3 services with outgoing connections from A) and the value for ADS is 2 (since there are 2 services with incoming connections to A). In general, we want these values to be low, as we want our services to have some connections but not be connected to all of the services, as this would defeat the purpose of the service architecture.

Within the created graph, we can also look at graph metrics. In this case we look at the **betweenness centrality score** and **cyclic dependencies**. The betweenness score is computed by taking the shortest path from every node to every other node in the graph, and looking at how often a service is included in all of these shortest paths. The more frequent this is, the more important the service is. The metric for cyclic dependencies is a boolean, this is 1 if the service is in a cycle within the graph and 0 when it is not in a cycle.

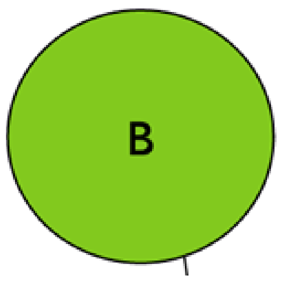
Finally, we want to determine the likelihood that services close to the investigated service, are also connected to the service. For this, we use a metric called **Local clustering coefficient** or **LCC** for short. The higher this value is, the more likely it is that neighbours of the investigated service are also connected.

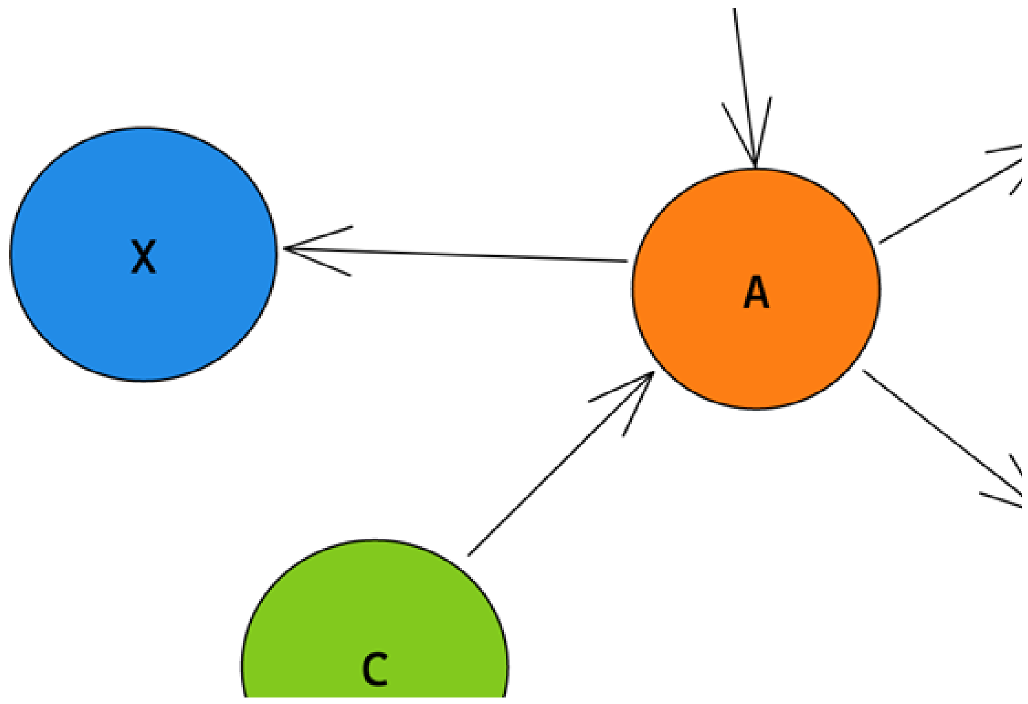
Source AIS/ADS: D. Rud, A. Schmietendorf and R. R. Dumke, Product Metrics for Service-Oriented Infrastructures, 2006, In IWSM/MetriKon.

Source Betweenness: U. BRANDES and C. PICH, "Centrality estimation in large networks," International Journal of Bifurcation and Chaos, vol. 17, no. 07, pp. 2303–2318, 2007. [Online]. Available: https://doi.org/10.1142/S0218127407018403

Source Cyclic dependency: A. M. Omer and A. Schill, "Automatic Management of Cyclic Dependency among Web Services," 2011 14th IEEE International Conference on Computational Science and Engineering, 2011, pp. 44-51, doi: 10.1109/CSE.2011.22.

Source LCC: Soffer, Sara Nadiv, and Alexei Vazquez. "Network clustering coefficient without degree-correlation biases." Physical Review E 71.5 (2005): 057101.




X
A
C

Metrics explanation (part 2 of 2)

In this section we look at the last 4 metrics that were used in the detection of anti-patterns.

Looking at the visual representation of a part of the service graph in the image, we can see service A again, like in the previous explanation. This time however, the red nodes represent the API endpoints that service A has.

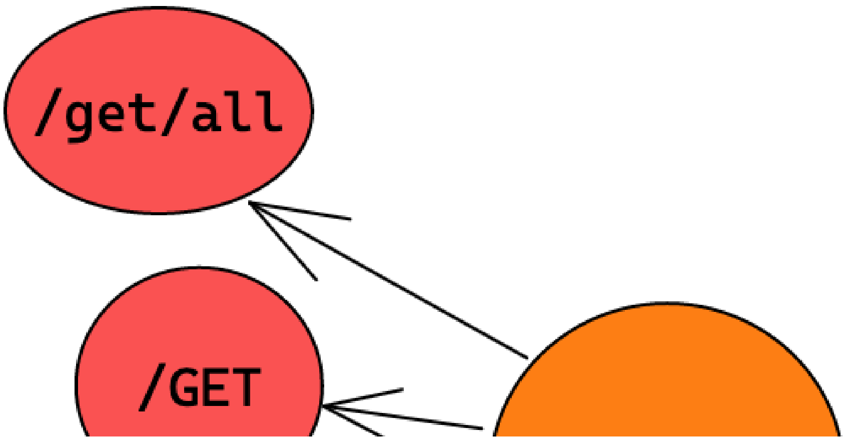
This is another metric that is used in the detection, we call this the **Weighted Service Interface Count** or **WSIC** for short. In this case, the value for **WSIC** for service A is 4, because there are 4 API endpoints. In general, we want this value to be low, as a service should not have too many responsibilities, otherwise a case could be made that the service should be split up.

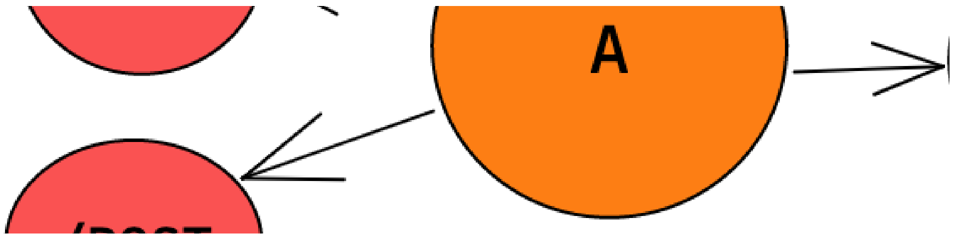
Combining the **WSIC** with the previously introduced **AIS** and **ADS** (which represented the number of incoming and outgoing service connections respectively), we can introduce the **Service Interface Usage Cohesion** or **SIUC**, which computes the cohesion of a service based on the number of operations invoked by other services. This is done by taking the sum of outgoing operation dependencies for all operations of A and dividing it by the sum of incoming operation dependencies for all operations of A, multiplied by the number of operations. For more information on this metric, feel free to consult the source paper listed below.

Finally we look at tracing messages. We take the average number of incoming messages and outgoing messages per hour for every service (the average is taken from a total of 12 hours). These messages are the same as those that can be found in

Source WSIC: M. Hirzalla, J. Cleland-Huang and A. Arsanjani, A Metrics Suite for Evaluating Flexibility and Complexity in Service Oriented Architectures, 2009, In Service-Oriented Computing – ICSOC 2008 Workshops: ICSOC 2008 International Workshops, Sydney, Australia, December 1st, 2008, Revised Selected Papers, Bernd J. Krämer, Kwei-Jay Lin, and Priya Narasimhan (Eds.). Lecture Notes in Computer Science, Vol. 4749. Springer Berlin Heidelberg, Berlin, Heidelberg, 41–52. DOI: https://doi.org/10.1007/978-3-642-01247-1_5

Source SIUC: M. Perepletchikov, C. Ryan and K. Frampton, Cohesion Metrics for Predicting Maintainability of Service-Oriented Software, 2007, In Seventh International Conference on Quality Software (QSIC 2007). IEEE, 328–335. DOI: https://doi.org/10.1109/QSIC.2007.4385516




A

Anti-patterns

Using these metrics, we attempt to detect anti-patterns within the service architecture at . The rest of the survey will consist of 4 anti-patterns, each will be explained. After this, 5 services are listed with their corresponding metrics in the acceptance environment.

One important note is that all of the metrics apart from the number of trace messages are **normalised**. This means that the values that will be displayed for the services are scaled based on the range of metrics that are present within the system. This scale ranges from 0 to 1, where 0 is the lowest value in the system and 1 being the highest.
For the number of trace messages, I will provide the average number of incoming/outgoing messages per service for comparison.

Feel free to ask questions if you are stuck or don't understand something. For a short summary of the metrics and the anti-patterns, have a look at the start page of Luduan (which can be found here (

Mega Service

When a service is flagged as a mega service, the service is considered to be too large or has too many responsibilities

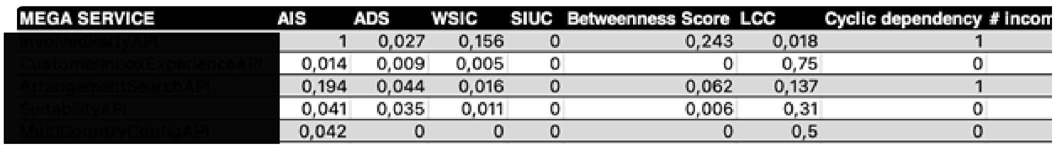
A way to resolve this anti-pattern is to split up the flagged service into smaller services

(summary of metrics and anti-patterns:

1

Would you classify the following services with the following metrics as anti-patterns, given the following metrics?

ais | Number of services which depend on a service
ads | Number of other services a service depends on
wsic | Number of API calls
siuc | Cohesion of a service based on the number of operations invoked per client
betweenness score | betweenness centrality score per service
lcc | the likelihood that neighbours of the service are also connected
cyclic dependency | 1 if the service is involved in a cyclic dependency, 0 otherwise.
On average, **94 messages** per hour per service are incoming and outgoing. *



MEGA SERVICE	AIS	ADS	WSIC	SIUC	Betweenness Score	LCC	Cyclic dependency	# incom
[redacted]	1	0,027	0,156	0	0,243	0,018	1	
[redacted]	0,014	0,009	0,005	0	0	0,75	0	
[redacted]	0,194	0,044	0,016	0	0,062	0,137	1	
[redacted]	0,041	0,035	0,011	0	0,006	0,31	0	
[redacted]	0,042	0	0	0	0	0,5	0	

Strongly disagree	Disagree	Neutral	Agree	Strongly Agree
○	○	○	○	○
○	○	○	○	○
○	○	○	○	○
○	○	○	○	○
○	○	○	○	○

2

How important were the different metrics for you to come to this conclusion? *

	Not important	Slightly important	Moderately important	Important	Very important
AIS	○	○	○	○	○
ADS	○	○	○	○	○
WSIC	○	○	○	○	○
SIUC	○	○	○	○	○
Betweenness score	○	○	○	○	○
LCC	○	○	○	○	○
Cyclic dependency	○	○	○	○	○
Number of incoming trace messages	○	○	○	○	○
Number of outgoing trace messages	○	○	○	○	○

3

Were there any other factors that influenced your decision on determining the likeliness of any service being classified as a mega service?

4

Do you have a relation to any of these services? *

- ☐ Yes, direct contributer
- ☐ Yes, consumer of one of the services
- ☐ I've heard of them
- ☐ No, nothing at all
- ☐ Other

Nano Service

When a service is flagged as a nano service, the service is considered to be too small, for example because it only has one subtask for another service.

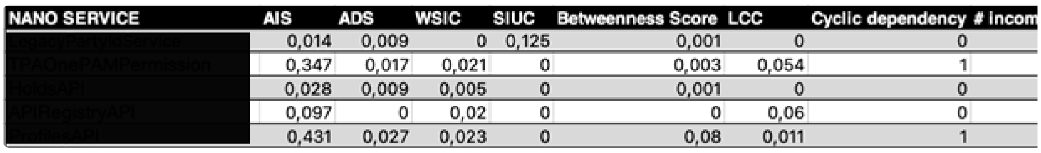
A way to resolve this anti-pattern is to merge the nano service into a nearby service to merge the responsibilities

(summary of metrics and anti-patterns:

5

Would you classify the following services with the following metrics as anti-patterns, given the following metrics?

ais | Number of services which depend on a service
ads | Number of other services a service depends on
wsic | Number of API calls
siuc | Cohesion of a service based on the number of operations invoked per client
betweenness score | betweenness centrality score per service
lcc | the likelihood that neighbours of the service are also connected
cyclic dependency | 1 if the service is involved in a cyclic dependency, 0 otherwise.
On average, **94 messages** per hour per service are incoming and outgoing.
*



NANO SERVICE	AIS	ADS	WSIC	SIUC	Betweenness Score	LCC	Cyclic dependency	# Incom
[redacted]	0,014	0,009	0	0,125	0,001	0	0	
[redacted]	0,347	0,017	0,021	0	0,003	0,054	1	
[redacted]	0,028	0,009	0,005	0	0,001	0	0	
[redacted]	0,097	0	0,02	0	0	0,06	0	
[redacted]	0,431	0,027	0,023	0	0,08	0,011	1	

Strongly disagree	Disagree	Neutral	Agree	Strongly Agree
○	○	○	○	○
○	○	○	○	○
○	○	○	○	○
○	○	○	○	○
○	○	○	○	○

6

How important were the different metrics for you to come to this conclusion? *

	Not important	Slightly important	Moderately important	Important	Very important
AIS	○	○	○	○	○
ADS	○	○	○	○	○
WSIC	○	○	○	○	○
SIUC	○	○	○	○	○
Betweenness score	○	○	○	○	○
LCC	○	○	○	○	○
Cyclic dependency	○	○	○	○	○
Number of incoming trace messages	○	○	○	○	○
Number of outgoing trace messages	○	○	○	○	○

7

Were there any other factors that influenced your decision on determining the likeliness of any service being classified as a nano service?

8

Do you have a relation to any of these services? *

- ☐ Yes, direct contributer
- ☐ Yes, consumer of one of the services
- ☐ I've heard of them
- ☐ No, nothing at all
- ☐ Other

Ambiguous Service

When a service is flagged as an ambiguous service, the service is considered to be generic and have a single operation to do, therefore making it "ambiguous" among the other services.

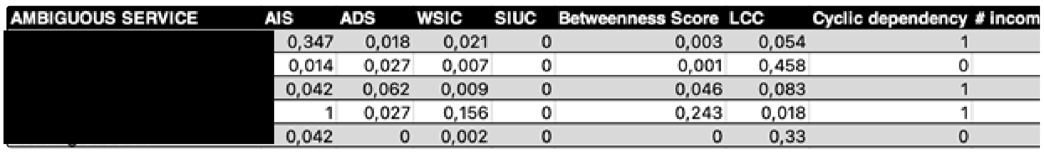
A way to resolve this anti-pattern is to split the single generic operation into multiple more specialised operations, to allow other services to request the information they require.

(summary of metrics and anti-patterns:

9

Would you classify the following services with the following metrics as anti-patterns, given the following metrics?

ais | Number of services which depend on a service
ads | Number of other services a service depends on
wsic | Number of API calls
siuc | Cohesion of a service based on the number of operations invoked per client
betweenness score | betweenness centrality score per service
lcc | the likelihood that neighbours of the service are also connected
cyclic dependency | 1 if the service is involved in a cyclic dependency, 0 otherwise.
On average, **94 messages** per hour per service are incoming and outgoing.
*



AMBIGUOUS SERVICE	AIS	ADS	WSIC	SIUC	Betweenness Score	LCC	Cyclic dependency	# incom
[redacted]	0,347	0,018	0,021	0	0,003	0,054	1	
[redacted]	0,014	0,027	0,007	0	0,001	0,458	0	
[redacted]	0,042	0,062	0,009	0	0,046	0,083	1	
[redacted]	1	0,027	0,156	0	0,243	0,018	1	
[redacted]	0,042	0	0,002	0	0	0,33	0	

Strongly disagree	Disagree	Neutral	Agree	Strongly Agree
○	○	○	○	○
○	○	○	○	○
○	○	○	○	○
○	○	○	○	○
○	○	○	○	○

10

How important were the different metrics for you to come to this conclusion? *

	Not important	Slightly important	Moderately important	Important	Very important
AIS	○	○	○	○	○
ADS	○	○	○	○	○
WSIC	○	○	○	○	○
SIUC	○	○	○	○	○
Betweenness score	○	○	○	○	○
LCC	○	○	○	○	○
Cyclic dependency	○	○	○	○	○
Number of incoming trace messages	○	○	○	○	○
Number of outgoing trace messages	○	○	○	○	○

11

Were there any other factors that influenced your decision on determining the likeliness of any service being classified as an ambiguous service?

12

Do you have a relation to any of these services? *

- ☐ Yes, direct contributer
- ☐ Yes, consumer of one of the services
- ☐ I've heard of them
- ☐ No, nothing at all
- ☐ Other

Bottleneck Service

When a service is flagged as a bottleneck service, this indicates that the service is being used by too many consumers and therefore will form a bottleneck and a single point of failure.

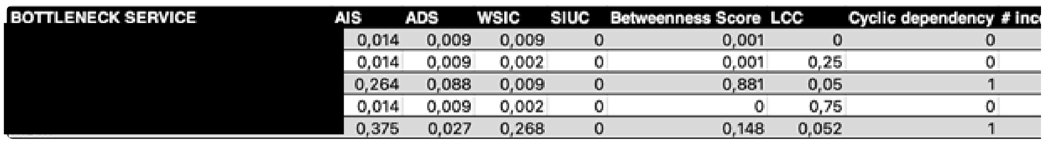
A way to resolve this anti-pattern is by splitting up the service into smaller services for each specific domain

(summary of metrics and anti-patterns:

13

Would you classify the following services with the following metrics as anti-patterns, given the following metrics?

ais | Number of services which depend on a service
ads | Number of other services a service depends on
wsic | Number of API calls
siuc | Cohesion of a service based on the number of operations invoked per client
betweenness score | betweenness centrality score per service
lcc | the likelihood that neighbours of the service are also connected
cyclic dependency | 1 if the service is involved in a cyclic dependency, 0 otherwise.
On average, **94 messages** per hour per service are incoming and outgoing.
*



BOTTLENECK SERVICE	AIS	ADS	WSIC	SIUC	Betweenness Score	LCC	Cyclic dependency	# inc
[redacted]	0,014	0,009	0,009	0	0,001	0	0	
[redacted]	0,014	0,009	0,002	0	0,001	0,25	0	
[redacted]	0,264	0,088	0,009	0	0,881	0,05	1	
[redacted]	0,014	0,009	0,002	0	0	0,75	0	
[redacted]	0,375	0,027	0,268	0	0,148	0,052	1	

Strongly disagree	Disagree	Neutral	Agree	Strongly Agree
○	○	○	○	○
○	○	○	○	○
○	○	○	○	○
○	○	○	○	○
○	○	○	○	○

14

How important were the different metrics for you to come to this conclusion? *

	Not important	Slightly important	Moderately important	Important	Very important
AIS	○	○	○	○	○
ADS	○	○	○	○	○
WSIC	○	○	○	○	○
SIUC	○	○	○	○	○
Betweenness score	○	○	○	○	○
LCC	○	○	○	○	○
Cyclic dependency	○	○	○	○	○
Number of incoming trace messages	○	○	○	○	○
Number of outgoing trace messages	○	○	○	○	○

15

Were there any other factors that influenced your decision on determining the likeliness of any service being classified as a bottleneck service?

16

Do you have a relation to any of these services? *

- ☐ Yes, direct contributer
- ☐ Yes, consumer of one of the services
- ☐ I've heard of them
- ☐ No, nothing at all
- ☐ Other

Final question

17

What is your role/profession within ?

18

Do you have any questions/feedback?

Thank you for filling in this survey!

Your help is much appreciated!
Feel free to have a look at Luduan and give feedback if you have any! Currently it is deployed on DTA and can be found here:

19

Do you want to have a chance to win a gift card and/or keep up to date with the results of the survey? If so please fill in your e-mail in the field below (otherwise feel free to keep this field empty)

COLOPHON

This document was typeset using LaTeX. The document layout was generated using a manually edited version of the MSc Geomatics Thesis Template on Overleaf by Hugo Ledoux[1], which is based on the `arsclassica` package by Lorenzo Pantieri, which is an adaption of the original `classicthesis` package from André Miede.

1 https://www.overleaf.com/latex/templates/msc-geomatics-thesis-template-tu-delft/yvjpkwvtkrwz