Using topology complexity as a software architecture design quality assessment

Kyran van der Laan
kyransem@gmail.com

Juli, 2023, 48 pages

Academic supervisor: Dr Z. (Zhiming) Zhao, Z.Zhao@uva.nl
Daily supervisor: Jan-Jelle Kester, Jan-Jelle.Kester@infosupport.com
Host organisation: Info Support, https://www.infosupport.com/

Universiteit van Amsterdam
Faculteit der Natuurwetenschappen, Wiskunde en Informatica
Master Software Engineering
http://www.software-engineering-amsterdam.nl

Abstract

The quality of a software architecture is very important for the success of a project. Wrong architectures will result in unnecessary time spent refactoring components and can even result in the failure of a system or project in the worst case. Research shows the software industry is making a migration from a monolithic architecture to the usage of microservices. This shift brings along the problem which is finding the correct granularity for the amount of microservices and how much work they should do individually. This problem is currently often handled by creating services as one sees fit at that moment which can result in the need to refactor services further on when they are incorrectly defined. In this research, we will investigate the main research question *How to effectively assess the design quality of software architecture from the topology of system components?* Current research shows various formulas for calculating the complexity of architecture given certain partitions but does not yet contain a clear comparison between these methods given this context. To these existing measures, we add a not yet tested formula that was created within the research company and stems from industry experience. We test the effectiveness of different formulas in their ability to assess architecture design quality using complexity as an indicator. Such a comparison between formulas has not been done in related research and will give further insight into a new possibility to use these formulas. We test the accuracy of formulas by creating a baseline measurement from interviews with professionals in the field about various architecture topologies. We then compare the formulas to this baseline measurement to assess their accuracy in representing these perceived best-quality architecture topologies. Using this method we were able to determine that the formula from industry practice is the best at calculating the optimal architecture topology whereas a clustering formula from related work is best at representing the total overall opinion of experts in the field. This means different formulas can be used in different situations depending on the context.

Contents

Chapter 1

Introduction

Microservice architecture is becoming more and more popular with many people stating it can aid in things such as continuous development and deliverance within a company [1]. However, there is still a lot of debate on how many services and how small services a system should be split up. Developing a system that consists purely of microservices will decrease the complexity of the individual components but has the possibility to greatly increase the complexity of the system overall. In that case, you are only moving the complexity of the system from inside the services to the outside, which often results in even harder to maintain systems since the complexity is now in between components [2]. On the other hand, keeping a system completely monolithic isn't the best solution either. The problems that arise when working with microservices are very concurrent since research shows that there is an active movement within the industry to use more microservice- systems [3]. Research shows that poor design quality and incorrect granularity of a system can result in lower quality software, increased difficulty to manage dependencies across services and a higher maintenance cost [4]. With more dependencies comes the cost of communication between services or functional units. This increased communication can lead to things such as more time needed to synchronise data or increased communication between teams. These disadvantageous costs can be mitigated or even avoided with a more optimal service granularity.

This thesis will focus on the complexity of software consisting of connected services. Connected services in this context are primarily aimed at microservices within a system. We will also focus on a more abstract meaning of connected services, namely "blocks of functionality" which can be grouped into microservices with our research. The current research already contains certain findings about the optimum granularity of a microservice system [5]. These findings include that Domain Driven Design (DDD) could be used to find the optimal granularity but don't provide a general calculation that can be done on such a system to test whether the split is optimal. Rather, multiple guidelines on how to divide a domain-driven design system are given. Our research elaborates on current literature with a formula not yet tested, henceforth referred to as the industry formula on account of it being created using industry practices within the research company, that can be used on a system to calculate if the overall complexity is low enough in comparison to other partitions of the same system. Furthermore, based on a conducted literature review, we can summarize a few insights into the problem from the already existing work. In this research, we will make a comparison between formulas, including the mentioned industry formula, in the effectiveness of representing high-quality architectural granularity as perceived by experienced individuals in the field.

By simply creating microservice systems, you don't necessarily bring down the complexity of the system as a whole [2]. This in itself already shows why this problem exists. Furthermore, even though it has been shown that microservice architecture is used more and more [3], a lot of developers are still uncertain about the usage of microservices [1]. This means that if you cannot find a good split between monolithic and microservice, those doubts will be confirmed which doesn't help the developmental process. There are several useful studies done on the complexity of software in itself accompanied by studies that analyse those complexity measures [6][7]. Our study isn't about measuring the complexity of the software itself but rather about finding a good granularity of the number of microservices. This means possible outcomes include a fully monolithic with no split in microservices, a maximum amount of microservices at the lowest level with each service containing the minimum amount of functionality and lastly, everything in between.

This problem arises at Info Support, the company providing the research environment, when they work on projects with clients and they need to develop a system that makes use of a microservice

architecture. It can also occur when the company is hired to advise on how to improve an already existing system to, for example, improve maintainability.

1.1 Problem statement

From this context, we can conclude that a balance must be found between keeping components monolithic and splitting up other parts into their respective microservices. Our research will address the aforementioned problem by comparing current methods of measuring the complexity of a system design including the not yet tested industry formula. As mentioned earlier, the reason that it is important to find a good split between microservice and monolithic is that not doing so will result in a project or organisational problem further on. This is backed up by literature such as in [4], in which the learnings of migrating from monolithic to microservices in a real-world application are described. This experience report shows that, in this case, microservices were created one at a time with little thought given to the eventual system overall. This resulted in microservices needing to be reworked such as splitting up or merging further down the line which resulted in valuable time lost. This shows that finding an optimum at the time of the creation of the system architecture will help with preventing such problems.

With our research aiming to be used at the time of the creation of the system architecture, it is important to keep in mind what information is usually available at that time. The most important things that need to be known in order for our research to be applicable, are the system architecture in terms of aggregates or blocks of functionality and the respective data flows or dependencies between them. This means you need to have the necessary requirements and documentation of the workings of your system before you are able to create a possible system architecture overview. Following this, due to the abstract nature of our research and architectures, there is the possibility to easily substitute things such as nodes and edges in the given architectural designs. The industry formula will still hold in those cases as this does not take context into account. This will add to the usability of this method as it is usable at various stages of context in the system life cycle.

This research is meant to be used by software architects that want to find a theoretical optimum granularity for their architecture. This means that, in practice, the architect will always have the necessary context of the system. A simple use case of this would be an architect that has created an architecture with a certain partitioning of microservices within said architecture. If the architect then wants to know whether a certain other split is better, this person can then use the best formula given in this research to figure out what partition would be best.

This research will not entirely solve the problem of finding an optimal granularity of microservices as this requires more than only one formula. In creating such an optimum you would also need to take team size, dynamics within teams and organisational size, amongst other things, into account. Rather, with the help of the industry formula, this research will add to the current literature and in turn help with the overall understanding of how to create a good split between monolithic and microservice.

1.1.1 Research questions

In order to solve this problem, we formulate the following main research question that we will answer in this thesis:

> ***RQ:*** *How to effectively assess the design quality of software architecture from the topology of system components?*

By looking at the topology of system components expressed in partitions containing nodes and edges, we are able to calculate a number on how complex said topology is. Complexity in itself relies on numerous things besides system architecture alone. Therefore, we aim to compare existing complexity measures to evaluate their accuracy rather than look at all the different aspects that come with a complete complexity of a system. Our research will provide insight into whether a certain partition granularity of a system architecture is less complex, and thus of higher quality, in comparison to other possible partitions of the same system. It is because of this that the calculated complexity values do not have a specific metric but are instead to be interpreted in comparison to the outcomes of possible other partitions of the same system. This fixes an important problem when it comes to comparing systems to each other. The formula outcome highly depends on the graph which means different interpretations of the functionality of a system will result in different graphs and in turn will result in different complexity numbers. This creates a situation in which it is almost impossible to compare systems if the graphs aren't created exactly equally. This is near impossible as no two software engineers think exactly alike.

Created graphs based on system documentation might get close but even one data edge can influence the eventual outcome. This means comparing systems to other systems when measuring complexity is prone to bias and faults. By comparing the possible system partitions to itself, the graph stays consistent and the complexity measurements are accurate within the same system. This also means the industry formula is not primarily meant to be used to compare different systems against each other.

The complexity part of the research question therefore mainly relies on the architecture with different partitions consisting of nodes and edges. We will focus on systems with architectures ranging from fully monolithic to as small micro-serviced as possible and the possible partitions in between. As shown before, the quality of the architecture design will aid in the maintainability of the system and can help prevent possible refactoring of the system due to poorly chosen partitions earlier on.

Sub-questions

We will use a few sub-questions to aid in the process of answering the aforementioned main research question and help increase the credibility of our findings. These sub-questions were created by dissecting the main research question into parts that all need to be answered. This means that the main research question can be answered by answering the sub-questions. These questions are:

1. *What are existing approaches for measuring architecture topology complexity?*
2. *How is topology complexity correlated with the design quality?*
3. *How to assess the architecture quality using the indicator of complexity?*

The goal of this thesis is to help with improving the process of transitioning from monolithic to micro-serviced as well as building a microservice system from the ground up. Our deliverable is an assessment of current formulas for measuring granularity complexity, including the industry formula. We will also deliver a reasoning on when to use what method including a basic plan on how you can use these findings in practice.

1.1.2 Research method

We will answer the research questions by first conducting a literature review in order to gain insight into the current state of the art practices. This will provide us with the gap in current literature that our research aims to fill as well as answer the first sub-question. It is important to first show the current work in order to give the ability to compare the industry formula to other formulas.

After having conducted the related work study, we will focus on analysing both real and fictitious systems with the formulas. We do this by first explaining the exact formulas in section 2. The original not yet tested formula was proposed by Roger Sessions in the white paper titled "The Mathematics of IT Simplification"[8]. This formula has then been altered by Raimond Brookman[1] at the research company in order to make it applicable for specific application to software systems using aggregates from the DDD principles. This also allows us to test the formula on other systems comprised of blocks of functionality as opposed to only the type of systems proposed by Roger Sessions. As stated before, the original formula and alteration can be found in chapters 2 and 3 respectively.

We will test the effectiveness of the formulas by presenting created system partitions to professionals and experienced workers within the field of microservice architecture systems. The architectures include a range of possibilities with monolithic and maximum amount of microservices always being included in the possible partitions. The other partitions were created and tested using the opinion of the professionals. We will interview these experienced individuals to gain an understanding of their perceived highest-quality system layout. With the knowledge gained from these professionals, we can find the most optimal formula for measuring the architecture design quality using complexity values. A continuation is an alteration to current methods to give an accurate representation of the perceived optimums. By proposing an alteration to the current methods to conform to the ideas of professionals about real systems, we are able to provide a realistic system partition. It should therefore be kept in mind that these formulas is not a one solves all solution but rather a tool useful in the creation of such systems.

By conducting the interviews, we aim to bridge the gap between theoretical and practical perceived complexity. This is important because this will give a more realistic view of how an optimum partition might look in practice which in turn will help with the problems as mentioned earlier in section 1.1.

After creating the professional opinion measurement, we will compare the calculated optimum system partition created by the formulas in order to gain insight into their effectiveness. By gathering the

[1] `https://blogs.infosupport.com/productivity-and-cost-effectiveness-with-ddd-defying-the-microservices-deathstar/`

perceptions of experienced people we can argue about the quality of certain architectural designs. This then allows us to compare the different algorithms against this perception in order to show what approach gives the highest quality architectural design. From this, we are able to show the differences between the industry formula and other algorithms and argue when you should use what method. This in turn allows our research to be used to determine what formula and method is applicable to the specific situation of the reader.

1.2 Contributions

Our research makes the following contributions:

1. Insight into the current state of the art practises for calculating complexity based on knowledge graphs.
2. A comparison of different methods and the industry formula in how representative they are of professional opinion and finding the optimal partitioning.
3. A validation and recommendation for improvement to current methods for calculating the optimal system partition granularity.
4. A rationale on what method to choose in what situation based on the made comparisons.

1.3 Outline

Chapter 2, contains the related work to this thesis and gives an outline of the current state of the art practises and the gap in literature our research aims to fill. Chapter 3 describes the original formula of Roger Sessions and the alteration made from industry practice. It also contains the conducted interviews with professionals that give insight into the perceived least complex system partitions. Lastly, this chapter will make comparisons between the industry formula and other algorithms and methods doing more or less the same to test the effectiveness of said methods. The results of our study are shown in Chapter 4 and are discussed in Chapter 5. Finally, we present our conclusion in Chapter 6 and end this research with chapter 7 containing future work.

Chapter 2

Background and Related work

In this chapter, we will present the previous work related to our research consisting of the background and the state of the art. The background will provide some necessary background information for this thesis. The state of the art will show current practices when it comes to the needed knowledge and the currently existing methods. Both sections will include a concluding subsection with the latter answering our first sub-question. A concise display of the subjects of each paper can be found in section 2.4 at the end of this chapter. Note some papers overlap in subjects which will be displayed in the aforementioned section in table 2.1. This table also contains the papers from section 2.1 as some papers also overlap in subject with the related work.

This literature research was conducted by selecting papers that argue in favour or against certain complexity metrics in order to gain insight into the different methods. The selection criteria for these papers include the ability to calculate a complexity measure given an architecture with certain partitions. This is in line with our main research question which aims to find an effective way to assess design quality based on the topology or architecture partitioning of a system. The method or formula suggested in the selected papers needs to have the ability to take partitioning into account as this allows us to create experiments where we compare against a measured optimal topology. One criterion which excludes a found method from the experiments in this paper is the need for system specifics such as weighed edges. This is because we are only interested in methods and formulas that do not have such constraints. As mentioned in section 2.1.2, such specifics are not always known yet and the methods in this research aim to be broadly usable at every development stage of a system. This includes the start of a project where an architecture could be created based on gathered requirements but exact weight values are not known yet. Lastly, an overview of the different methods of measuring graph complexity that are in line with our experiments will be given at the end of this chapter.

2.1 Background

This section will present the necessary background information for this thesis. We will focus on how to define microservice systems as well as how to represent system architectures as graphs. The section will finish with a short summary about the background. The material discussed was found by the usage of Google Scholar and the University of Amsterdam (UVA) online library. Physical books, if applicable, were largely provided by Info Support, the company providing the research environment.

2.1.1 Defining microservice systems

Defining and representing systems ranging from monolithic to microservice architecture is important for our research because it forms the base of our experiments. Without properly creating examples and finding suitable system representations, we won't be able to argue about the statistical significance of our results.

The first paper, titled "Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud"[9], contains research that has been conducted into companies that currently use microservice architectures and what that means in those contexts. This paper gives examples of how a system might look if it was implemented using a monolithic system as well as using a microservice architecture. Most importantly, this research shows a cost analysis to further the argument of switching from monolithic to microservices. This analysis shows that for a small system, the total costs

per month come out to $65,56 cheaper for a microservice implementation using Amazon Web Services (AWS) [9]. This shows that the costs for a large system used in real business can benefit substantially from using microservices. Another finding in this paper highlights another benefit of using microservice architecture. This benefit namely consists of the ability to easily deploy smaller parts of a large system. This can be advantageous because it allows for easier and quicker development of features or maintaining a system on account of not needing to deploy an entire system at once when new code is introduced. These findings give a good understanding of the differences between monolithic and microservice architecture and show why this research can be beneficial as we aim to help with the process of going from monolithic to microservices. Similarly, this shows why our research can help with the current way of working as it will help gain a better understanding of the system at hand.

Important for our research are the methods of splitting a system into individual services. Research shows there are various methods of creating such a split including splitting by use case, splitting by verbs (think different subsystems), splitting by resources and using the Single Responsible Principle [10]. These methods allow for the creation of microservices given certain levels of abstraction. This can be utilized in this research to create example architectures for the experiments if needed.

With the earlier mentioned upsides to using microservices, it is also important to look at possible negative effects of using this architectural design. One of the most important possible drawbacks is the fact that microservices don't necessarily bring the overall complexity of a system down as opposed to a monolithic system [2]. This is explained in the article "Microservices, a definition of this new architectural term"[2] in which the authors speak about software components expressed in microservices: "Another issue is If the components do not compose cleanly, then all you are doing is shifting complexity from inside a component to the connections between components. Not just does this just move complexity around, it moves it to a place that's less explicit and harder to control." [2]. The main takeaway to gather from this is that it shows why defining microservices should be done with care and reasoning. The aim of this research is to help with these issues where you are merely shifting complexity around by giving software architects the ability to calculate the complexity of possible architectural designs at the moment of creation.

2.1.2 Representing system architectures as graphs

This research focuses on the usage of systems represented as graphs. These graphs consist of system functionalities as nodes, such as aggregates as defined by DDD, and data flows such as calls or dependencies between nodes represented as edges. It is therefore imperative that the current way of creating such graphs is explained clearly.

One way of creating such "knowledge graphs" is by extracting different nodes using documented requirements and decisions made during the development of the monolithic application [11]. These nodes can be defined as module nodes, function nodes, data nodes and resource nodes which all refer to their respective type of requirement or decision [11]. After this, different types of relations between the nodes are extracted in order to create the necessary edges between the nodes. The added benefit to using this method for creating knowledge graphs is that it requires relatively low experience with microservice creation. A necessary requirement of this method is the addition of edge weights in the created graph. The purpose of these weights is further explained in section 2.2. These edge weights could be considered a drawback to using this method as these specifics are not always known yet, especially at the start of a project. Our research will therefore experiment on methods where such specifics are not known yet or are not needed in order to make the research applicable to a wider range of situations.

Similar research to the method mentioned earlier[11] shows another approach in terms of node extraction and used algorithm. In the paper "Extraction of microservices from monolithic software architectures"[12], the code base of an application is used to extract nodes and create the knowledge graph. By analysing code specifics such as classes and calls this method creates an initial knowledge graph which is then used to create the microservice architecture. The algorithm in this paper uses a minimal spanning tree and the deletion of edges in order to iteratively calculate the optimal microservice partitioning. Edges are deleted until a set parameter is reached because autonomously determining when to stop is not possible with this method. The created system is evaluated by comparing the performance of the created system in terms of execution time. Further metrics used in this paper to evaluate the system are coupling, time size reduction and average domain redundancy. The method described in this paper once again uses edge weights for the creation of microservices. This means this method is also not suitable for the experiments proposed in our research.

Where our research differs further is that we do not necessarily aim to exclusively create microservices

as described in the mentioned methods [11][12], but rather create autonomy boundaries which can be interpreted as more things than just deployment boundaries for microservices. One of these things could be the possibility to decide what features to couple and assign to certain teams. By researching methods that do not require edge weights our research can be used more easily and earlier on with less knowledge about the system at hand. This in turn means both researches can be used at different stages of system development. One last difference is that our research is to be used to validate found partitions of the system against other possible partitions of the same system. This means we can more easily provide a range of possible partitions in the same complexity range rather than give one possible partition such as in this research.

2.1.3 Summary

The current perceived practice in the industry in creating and defining microservice architectures is by using various methods to define strongly coupled code. This is often done by looking at the functionality of services or features and seeing what can be combined into a microservice. It has been shown that creating microservices is a difficult process which can result in merely moving complexity or making wrong assumptions at the time of creation. This shows a research possibility for improving the process at the time of creation of system architectures. Our research can be used at the moment of creation or when migrating from a monolithic system and can help with preventing making wrong assumptions about what system functionalities to group into partitions and eventually microservices.

Alongside this, the current trend in creating system architecture graphs can be summarized as extracting nodes and relations based on system documentation as well as code bases. After the initial creation, most graphs require further work in order to remove redundancy. Most works agree that the quality of the complexity assessment relies on the ability of the engineers to create the right knowledge graph, especially in the case of using prior created documentation. Some mentioned methods for testing the effectiveness of the created architectures include cohesion, coupling and time size reduction.

2.2 State of the art

2.2.1 Current situation

This section provides insight into the current situation when it comes to creating, measuring and working with connected service systems. We aim to show what the current perceived problems from the industry are which allow us to argue about why our research adds value.

A qualitative study done with experts about the current state and views of working with microservices discusses the benefits and issues with creating a microservice architecture system [1]. The most important findings within the issues are that most architects have a lot of uncertainty with using microservice architecture. Another important finding is that microservices may not always be the best option for a company depending on the type of company. Companies that see IT as “a necessary evil” [1] and large record databases of big companies do not benefit from the microservice architecture as per the views presented in this paper. What this shows is that microservice architecture in itself might not always be the best option which is important to keep in mind for this research. Our research includes a fully monolithic system as one of the possibilities in the experiments which allows the mentioned type of companies to get insight into whether microservices or monolithic would be the best option in their case.

One very important finding is the fact that the current industry seems to be moving from Software-Oriented architecture to microservices [3]. The main reasoning for this ties in with a benefit of microservices earlier mentioned in section 2.1 with it being that microservice architecture has the ability to independently deploy services and has elastic scalability. These factors contribute to the ease of usability which in turn shows why the industry is moving towards the usage of microservices. These findings show that our research is contemporary because we aim to help with the creation of microservice systems. As seen earlier, developers currently have some doubts about using microservice architecture. This is also contributed to the fact that services are often not classified correctly the first time.

This incorrect classification is highlighted in a real-life report that displays the process of migrating a system from monolithic to microservices [4]. An important insight from the described process in this report is that the microservices were created one at a time with the grouping of functionalities being decided at the time of creation. As described, this resulted in more changes further on in the project when they found some functionalities needed to be split and some needed to be grouped. This creates unnecessary time spent on refactoring said services which could have been used elsewhere if the services

were identified correctly the first time. This research will give the possibility to do so with the help of the most optimal formulas for classifying such groupings of functionalities.

Lastly, because the model from industry practice is based on the DDD principles, it is important to show why this is a good practice to start with. In a paper titled "Does Domain-Driven Design Lead to Finding the Optimal Modularity of a Microservice?" [5], the author shows that DDD can be used to create an optimum in the granularity of a microservice system. The author uses multiple ways to define a system using DDD and makes calculations based on the used design ways. The main finding and takeaway of this paper show that DDD can indeed be used to find a theoretical optimum. What this means is that the formula created from industry practice is a good starting point to be tested for accuracy as it originates from an alteration to a formula to conform to the DDD practices.

2.2.2 Current methods of measuring architecture complexity

It is vital for our research to include the current and different state of the art practices in measuring the complexity of partitioned architectures. Without this, we are not able to show our work contributes to this concurrent issue and adds value to the existing literature.

One such method that is state of the art but requires weighted edges is the Louvain method [13] as used in earlier cited work [11]. The Louvain method creates "communities" of strongly coupled nodes which can then be translated into microservices. This method has been proven to create fairly good performing microservices regarding cohesion, coupling, code redundancy rate and team size reduction [11]. It also has the added benefit that such an algorithm requires little knowledge about microservices as it creates these communities automatically. However, as stated before edge weights are not always a given and incorrect or biased weight may skew the results to an unwanted system architecture.

Arguably the most important paper for this research is the white paper called "The Mathematics of IT Simplification"[8]. In this paper, the author proposes a formula for calculating the complexity of a system comprised of modules and data connections. The importance of this paper stems from the fact that the formula in this paper is the original formula that Raimond Brookman altered to create the industry formula. For the exact alterations and complete industry formula refer to section 3.3. The original formula as proposed in this paper for the total complexity of a system consisting of partitions is the following:

$$C(Architecture) = \sum_{i=1}^{m} (10^{3.1 \times log(M_i)} + 10^{3.1 \times log(N_i)})$$

Here, m represents the different modules or partitions, M is the amount of coordination dependencies to other modules or partitions and N is the number of functions within the partition. Do note the original drive containing the white paper is no longer functioning, the version used in this research can be found in a snapshot of said drive[1].

Further research shows similar formulas for calculating the complexity value of an architecture given a certain topology. Earlier mentioned methods of measuring the quality of created architectures included coupling and cohesion. This is further backed up by research that gives an extensive overview of possible complexity measures for architectures ranging from monolithic to microservices [14]. This paper argues that the most important evaluations that can be used for measuring the complexity of a microservice are cohesion and coupling. Cohesion is defined as the grouping of entities and whether each entity in a group is accessed when one of the entities is accessed. The cohesion metric in this paper is split up into the cohesion of a cluster, which refers to a cluster of functionality, and the cohesion of a decomposition, which is the average cohesion of a group of clusters. The second metric given is coupling. Coupling in this context refers to the distance between groups of entities. This is shown to be used to determine how isolated microservices are. This measure is again split up into the coupling of a cluster and the coupling of a decomposition. According to this source [14], it stands to reason that the coupling metric does not work on fully monolithic systems as there is no other cluster to measure the coupling between.

These cohesion and coupling measures can however also be used as formulas for calculating the initial complexity of a topology. The paper titled "Measuring graph abstractions of software: an information-theory approach"[15] provides, amongst other things, the formulas for coupling and cohesion to measure the complexity given the partitioning of a topology. A general complexity formula is used to determine the total complexity of a topology and is used in the formulas for cohesion and coupling. The general complexity of an entire system S and the complexity of a partition m_k of S both with n nodes are defined

[1] http://web.archive.org/web/20160115041915/https://dl.dropboxusercontent.com/u/97323460/WebDocuments/WhitePapers/MathOfITSimplification-103.pdf

respectively as:

$$Complexity(S) = (\sum_{i=1}^{n} Size(S_i)) - Size(S^{\#})$$

$$Complexity(m_k|S) = (\sum_{i \in m_k} Size(S_i)) - Size(m_k|S^{\#})$$

The formula $Size(S)$, in which S can be substituted for subsets of S such as S_i, is given as:

$$Size(S) = \sum_{i=1}^{n} (-\log_2 p_L(i))$$

Additionally, S_i means the node edge matrix of node i and all its connections and $S^{\#}$ stands for the system graph with all unreachable nodes deleted (e.g. nodes without any edges). The $p_L(i)$ function represents the probability mass function expressed in the proportion of the row pattern on row i in respect to other rows in the node edge matrix. This paper then defines the formulas for the coupling and cohesion metrics. The coupling complexity metric of system S measures the system divided into modules. This system is represented as MS and the metric is defined as:

$$Coupling(MS) = Complexity(MS^{*})$$

The * in this formula stands for the intermodule edges graph. This means that only the nodes and edges within modules are counted. Opposite of this is the intramodule edges graph. This is represented as $^\circ$ and consists only of the nodes whose edges cross module boundaries and the edges themselves. Lastly, $MS^{(n)}$ is defined as "consisting of all the nodes in MS and all the possible edges between those nodes and let all nodes be in one module"[15]. With these definitions, the cohesion complexity metric is defined in this paper as:

$$Cohesion(MS) = \frac{Complexity(MS^{\circ})}{Complexity(MS^{(n)})}$$

These measurements are in line with the not yet tested industry formula and calculate complexity values of a topology with different partitions. This means these formulas are perfectly suited to use in the comparison of how accurately different methods reflect a measured optimal architecture topology. By creating example systems and partitions, we are able to rank each possible partitioning of a said system using different formulas and methods. We are then able to compare this to the outcomes of the interviews in order to review how each method compares to the real perceived complexity.

Further research shows one more formula that has the ability to calculate a complexity value given a graph with partitions. This formula uses the different partitions in order to calculate the total complexity. The formula proposed in this paper [16] will henceforth be referred to as the cluster or clustering formula as it has no apparent given name. The exact formula is as follows:

$$C = \sum_{j=1}^{K} (n_j e_j) + NE_0$$

In this formula, K represents the total amount of partitions with each individual partition defined as j. $n_j e_j$ are the number of nodes and edges contained within partition j respectively. Lastly, N is the total amount of nodes in the entire system and E_0 is the amount of edges between partitions. This paper [16] shows that using this complexity measurement allows for the minimizing of clusters which in turn helps optimise the system. This clustering formula is also suited to be included in the experiment as with the coupling and cohesion formulas for the same reason.

With the state of the art, it is also important to look at negative results for methods. One very popular method that has been used to gauge software complexity is cyclomatic complexity. This metric uses individual paths of a program such as calls and statements to assess the complexity of the software. It has been shown that this complexity however might not be the best possible metric. Research shows that the original model for cyclomatic complexity has never been standardized which leads to people interpreting and implementing this measure differently [17]. This creates ambiguity about the measurement as a whole and can result in unfair comparisons between systems if people used different interpretations to calculate the cyclomatic complexity. Said research states that "the search for a general complexity metric based upon program properties is a futile task" [17]. It is argued that it might be a better

approach to go up one abstraction level and look at program properties based on the concept of software design. This ties into our research in that we assess a formula usable with the combination of abstract system representations and DDD. This also shows the importance of clearly defining measurements and explaining formulas.

When it comes to measuring the complexity of a system topology it has been shown to be beneficial to be used in combination with other complexity measures. In a paper titled "A qualitative method for measuring the structural complexity of software systems based on complex networks" [18], the author proposes a new method of measuring system complexity based on directed graphs with a combination of system entropy and edge influence. The paper shows how systems can be represented as directed graphs created based on component and connector plots. The author argues that such an approach to calculating the complexity of systems can prove to be useful on top of the standard complexity measures which often look an abstraction level deeper than the overall system structure. It is shown that this is beneficial because it adds to the aforementioned more specific complexity measures that often look at specific parts of systems. These findings are of direct influence on this research as it shows that the usage of system entropy and edge influence are viable options for calculating the complexity of a graph. This in turn shows the method used in this research is viable and possibly usable in real architectures. Where our research is novel compared to this paper in that we aim to give insight into the optimum division of autonomy boundaries or microservices whereas this paper only calculates a complexity measure based on the system as a whole.

In the specific case of calculating the optimal architecture granularity based on knowledge graphs and partitions, related work shows a couple of possible measures. From the found related work, we can shortly summarize the following metrics that evaluate the granularity of a graph partitioning:

1. **Cohesion**, which focuses on how focused classes within the code are. High cohesion means classes are focused on not having too broad functionalities and are generally considered to be better.
2. **Coupling**, which focuses on how related classes are to each other. Low coupling means classes can be changed more easily without affecting others and are considered to be better.
3. **Clustering**, which is a general clustering formula which also measures the quality of a certain architecture partition.

2.3 Concluding

From the gathered related work we can conclude that some work has already been done in the creation of microservice applications from monolithic systems and measuring the complexity of software architecture topology. We showed that there are various proposed solutions and approaches using different methods similar to our research. The main methods we found in the related work that assess certain partitions on a graph are:

1. **Cohesion**
2. **Coupling**
3. **Clustering**

To this list, the formula from industry practice is added as a fourth and final option. The first three methods are in line with the industry formula in that they measure the complexity of a graph given a certain partition. Furthermore, a lot of approaches give one possible solution and treat it as the theoretical best one without giving the possibility for a similar partition which may work better given some system context. We will compare the formula from industry practice against these popular complexity metrics. By doing this, we are able to compare our findings and assess the effectiveness of the different formulas.

The current state of the art practice related to our research is to create knowledge graphs and use an algorithm on said graph to create possible partitions. Related work shows that these graphs often require weighted edges in order to create suitable partitions[11][12]. Our research aims to fill the gap where such system specifics are not known yet. The process described in this thesis will be usable with minimal system feature descriptions in the form of a knowledge graph without weighed edges. This allows for the usage of our findings earlier on in the process of migrating a system from monolithic to microservice or even when creating a microservice architecture without having a running system prior. Similar research shows methods that can be used while migrating systems from monolithic to microservice, not while creating a microservice system from the ground up[15][16]. The proposed algorithms in these papers

could be used on knowledge graphs created without the usage of the described source code extraction, but that would require the architects to make the assumption that those methods work just as well in a situation different than substantiated in those papers.

In conclusion, this means that our research aims to fill the research gap which is a comparison between current methods and creating partitions with little prior system knowledge. We do this by comparing the industry practice formula that requires less system knowledge than other algorithms against existing formulas that also do not require these metrics. We will also address the gap that is to show when some formulas could be more advantageous over others as it is most likely that not one formula is always the best. From the literature, no concise comparison of these formulas and when to use what formula has been found.

This concludes in the answering of the first sub-question with the aforementioned three methods and the not yet tested formula from industry practice being the existing approaches for measuring architecture topology complexity. As stated, there exist more methods that give similar insights into the complexity but as shown, they often require system specifics such as edge weights. This means these methods will not be taken into consideration as we want formulas that are usable at any stage of development meaning few system specifics should be required to use said formulas.

For the second sub-question of an assessment between formulas in performance of reflecting accurate design quality, no work has been found. Each of the three related work formulas has been tested with its own range of assessments but not against each other. Our research will make a comparison between the three found existing methods as well as the not yet tested formula from industry practice.

2.4 Subject overview

A short overview of the covered subject by each paper cited in this chapter is given in table 2.1 in order of citation appearance.

Article	**Background**		**Current practises**	
	Representing systems	Defining microservice	Measuring complexity	Current situation
[1]				**X**
[2]	**X**	**X**		
[3]				**X**
[4]				**X**
[5]	**X**			**X**
[8]	**X**		**X**	
[9]	**X**	**X**		
[10]	**X**	**X**		
[11]	**X**			**X**
[12]	**X**			**X**
[14]		**X**	**X**	
[15]			**X**	
[16]	**X**		**X**	**X**
[17]			**X**	
[18]	**X**		**X**	

Table 2.1: Related work subject overview

Chapter 3

Research

3.1 Methodology

In this research, we will use the perception of experienced individuals to create a measurement against which we will measure the effectiveness of different formulas. The reason for doing so is that we identified a gap in bridging theoretical optimal architecture partitions versus perceived real-life optimums. We will start by conducting interviews by creating multiple architectures at various levels of abstraction. By interviewing multiple professionals we can create a general consensus on what is perceived as the highest-quality architectures. The result of this will be rankings for each architecture from best to worst partitions.

The general consensus of the interviews can be gathered in a few possible ways. The first way is by simply taking the average of indexes a partition was ranked at and creating a ranking based on that. This would not work however because you get unfair skewing towards outliers. This could also result in partitions with the same average ranking disregarding how sure the professionals are about said ranking which should also have a certain influence.

A second option is to choose the most occurring index at which a partition was ranked at and use that as the ranking for said partition. This would also not work because it completely disregards the other rankings for that partition. To show why this would not work consider the following example:

	Partition 1 ranked at index	Partition 2 ranked at index
Person 1	1	1
Person 2	1	1
Person 3	1	1
Person 4	4	2
Person 5	4	2
Person 6	5	2
Person 7	5	2

Table 3.1: Example ranking.

In table 3.1, each cell represents at which index the partition from that column was ranked. If you only take the most occurring value to decide the majority consensus, you completely ignore the rest of the opinions. In this example, partition 1 would be ranked first as the most occurring index is 1 and partition 2 would be ranked second. Despite this, partition 1 was ranked last and second to last more than half of the time by other interviewees and partition 2 was ranked at the first and second spot most often. Logically partition 2 should be considered better than partition 1 as it was ranked higher the majority of the time.

The solution we choose in this research to get the majority consensus is by creating boxplots and looking at the median values. By using the median instead of the average, the negative effects of outliers are mitigated and partitions with a divided ranking are no longer ranked before other rankings with stronger preferences. By using boxplots we are also able to argue which partitioning would be ranked

better in the case that two partitions have the same median. By comparing the maximum and minimum as well as the 25^{th} and 75^{th} percentiles, we can see which partitions still have slight preference despite having the same median.

After having created a ranking for each architecture from best to worst, we will calculate the complexity of said architectures using the four different formulas highlighted in section 3.2. By then creating a ranking of the same partitions using these formulas, we can compare what formula represents the professional opinion the best. The comparison of the formulas against the perceived optimal solution will be done in two parts.

The first part is using solely the ranking of the measured architectures to compare against the ranking of the formulas. This initial comparison will give an estimate of the ability of the formulas to represent a ranking in itself. After making this comparison we will be able to draw a preliminary conclusion about the formulas. Furthermore, we will not only compare the complete rankings but also the sole optimum and top 3. The ability of a formula to accurately represent only the optimal partition is valuable in itself as the number one partitioning is arguably the most important insight for architects using said formulas. This distinction between complete ranking and sole optimum is necessary as it is not a given that a formula is the best at finding the optimum partition if it has the closest overall ranking.

In the second part, in order to further our earlier findings based on only the rankings, we will investigate further into the rankings with the actual complexity values. This is important because if the calculated values of two partitions are very close together and are ranked at the wrong index, it is less bad than when two partitions with very different complexity values are ranked wrong. This comparison with the values will result in a stronger conclusion about the formulas. The exact way to calculate the differences using the complexity values is done in three steps:

Step one is to normalize all values within the rankings to be a number between zero and one for the best and worst partitions respectively. We normalize these complexity numbers because we are not only interested in the abstract ranking but also whether the actual complexity measures are close to each other. This is important to know because two partitions that have very similar complexity measures may be switched with little effect whereas two partitions with a great difference in complexity should never be switched within the ranking. For example, if a perceived optimum has a partition ranking of 1,2,3,4,5 and a calculated ranking is 1,2,4,3,5, then partition 3 and 4 are incorrectly ranked. However, if partitions 3 and 4 have very similar complexity values this incorrect ranking is less severe than when the calculated complexity of partitions 3 and 4 differ greatly. The formula used to normalize the complexity numbers is as follows:

$$Norm(i) = (i - Minimum(n))/(Maximum(n) - Minimum(n))$$

With i the value of one index within the ranking and n all the complexity values within the ranking. This will result in a value of 0 for the minimum complexity value, 1 for the maximum complexity value and distributed values between 0 and 1 for the values in between. One exception to this is for the Cohesion measurement which is the other way around. This is because a higher cohesion complexity value is better as opposed to worse for the other complexity metrics. This means that the highest cohesion value needs to be a normalized value of 0 and the lowest a normalized value of 1. We accomplish this by changing minimum into maximum and vice versa.

The second step is to calculate the distance vector of the ranking using the perceived optimum ranking. We do this by comparing the normalized value of the calculated complexity against the normalized value of where that partition actually should have been. This means that for a given partition i, the normalized value is compared against the normalized value of partition i in the actual perceived optimum. The exact formula given this explanation is the following:

$$Dist(n, m) = \sqrt{\sum_{i=1}^{n} (Norm(n_i) - Norm(m_i))^2}$$

Here, n is the ranking of the calculated complexity values, m is the actual perceived optimum ranking and i is the ranking index. This gives a total distance for one ranking against the perceived optimum. We also calculate the distance value of only the optimum partition because, next to the ability to accurately represent the total ranking, we are also interested in the ability to accurately calculate the sole optimum architecture partition.

The last step is calculating the averages for the total distance values across the measured architectures and the distance of optimums for the measured architectures. After gathering these values, we will analyse

them in order to gain insight into the performance of all formulas in representing the best partitioning. With this insight, we are able to argue what formula is best to use in what situation.

In the final part of this research, we will test our findings and insights about the formulas by using a real-life architecture and another ranking from professional opinion.

3.2 Current methods for measuring architecture complexity

As presented in related work, there exist a number of metrics for calculating the complexity of a graph consisting of one or more partitions. We use these metrics to compare against the professional opinion including the industry formula. To reiterate, these are the metrics:

1. Cohesion as described in [15]
2. Coupling as described in [15]
3. Clustering as described in [16]

For the exact formulas and an explanation of said formulas, please refer to section 2.2.2. Literature shows that both Cohesion and Coupling are state of the art metrics for measuring the complexity of partitioned graphs [15][14]. Furthermore, the clustering complexity as described in [16] also shows potential with it measuring different partitions and connections between them. This is advantageous because it is in line with the industry formula and attempts to measure the complexity similarly. It is for these reasons we decide to include these measurements in the comparison against the professional perception as described in section 3.4.

3.3 The industry formula

In this research, we assess a not yet tested formula for evaluating partitioned systems on top of the comparison between all existing formulas. The untested formula stems from a formula originally proposed in the white paper "The Mathematics of IT Simplification"[8]. This formula, as described in section 2.2.2, has been altered to conform to the DDD principles[1] with the usage of aggregates as nodes which is then referred to as the industry formula. The alterations were made by Raymond Brookman while he was employed at the research company Info Support. This means the alterations were made based on experience within said company. Since this formula has never been tested and has been altered, this is also a large part of our main contribution to the existing literature as it has never been done before with this formula.

The reason for the aforementioned alterations stems from the fact that the original formula is from 2011 and was no longer suitable for modern-day practices. As Raymond Brookman argued in his original alteration, the DDD principles are suitable to be usable in defining microservices. It is for this reason he chose to alter the original formula using these principles. In practice, the exact formula itself did not change but rather the definition of the variables used in the formula. The original formula was meant to be used to split up a project into modules. The new formula changes this to split up a topological view of a system into autonomy boundaries. This results in the industry formula for the complexity of an architecture with partitions as follows:

$$C(Architecture) = \sum_{i=1}^{m} (10^{3.1 \times log(bf_i)} + 10^{3.1 \times log(cn_i)})$$

Which can be simplified to:

$$C(Architecture) = \sum_{i=1}^{m} ({bf_i}^{3.1} + {cn_i}^{3.1})$$

Consisting of the following characteristics:

- Log is based 10 (for the original formula)
- i = A partition in the architecture
- m = All partitions
- bf_i = Number of nodes within a partition
- cn_i = Number of edges to other partitions, bidirectional

[1] https://blogs.infosupport.com/productivity-and-cost-effectiveness-with-ddd-defying-the-microservices-deathstar/

With not every architecture being created while keeping the DDD principles in mind, we redefine the formula to be more abstract. This means that nodes do not have to adhere to the exact definition of an aggregate and boundaries do not necessarily have to be autonomy boundaries. What this means exactly is that within the most abstract architectures, the nodes can represent either aggregates or system functionalities such as classes or blocks of code. The edges represent dependencies or data flows. This in turn means the edges can be interpreted as things such as API calls, consistent data between nodes or something like method calls between classes.

3.4 Professional perception

As mentioned before, in order to create a trustworthy criterion to compare the different algorithms, we have conducted interviews with architects and developers with considerable experience. A total of 10 individuals are interviewed with all interviewees having at least 5 and at most 20 years of experience with creating architectures including monolithic and microservices or working with these architectures and microservices and maintaining them.

In order to create a diverse measure, we included the following architectures:

- **Without context**
    1. Abstract fully connected, each node is connected to every other node, no directional edges.
    2. Abstract system, no directional edges.
    3. Abstract system, with directional edges.
- **With context**
    4. Real system, with directional edges.
    5. Real system, large architecture with directional edges.

These architectures, including their presented partitions, can be found in appendix section A with the list numbers corresponding to the architecture numbers respectively. Do note that architecture 5 does not contain context within this research. The context was presented to the professionals but cannot be included in this research with context due to confidentiality.

Along with these architectures, the interviewees were also presented with a sixth architecture which was the exact same as the architecture in A.4 but with added node labels. This architecture can be found in A.6 and was used to test the influence of context on the professional opinion and whether earlier choices were upheld with said context.

In a real situation, a fully connected architecture will rarely occur. Nevertheless, it is still insightful to gather information about such a system. As stated in chapter 1, abstract architectures can be substituted with the necessary interpretations at the time of creating the architecture. Each used architecture has 5 different system partitions which always contain a fully monolithic and an as small as reasonably possible micro-serviced partition as two of the options. These partitions are one of many possibilities and were chosen with the insight of a professional or based on a real system design. We chose very different partitioning designs for each architecture in order to capture a wide range of real-life possibilities. During the interviews, the interviewees were asked to give a ranking of the different system partitions according to what their perceived most realistic and best architecture was. This means the ranking they give ranges from best to worst architecture. Along with this, they were asked to give a reason for their ranking to further their argument.

It is very important to realise the scope of the interviews and thus the measure we compare against. The interviewees consisted of professionals with experience in numerous types of systems. These included systems such as user management with databases and functionalities up to complete production systems with multiple business functionalities. Therefore, we instructed the interviewed professionals to reason from the viewpoint of their field of expertise in order to gain insight from different fields of view.

Chapter 4

Results

This chapter will highlight the outcomes of the interviews and compare the formulas against these results. This also includes the rationale for choices made by professionals to support their ranking.

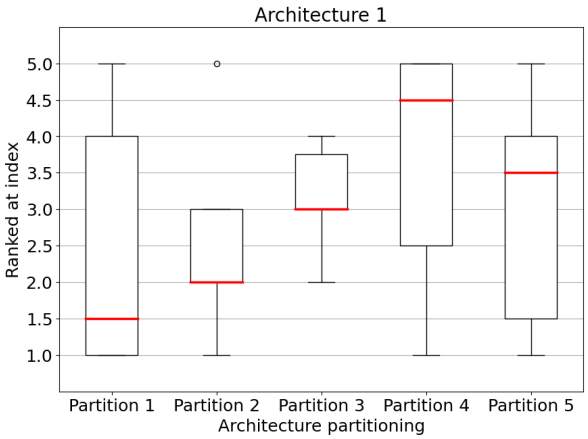
4.1 Professional perception outcomes

The first insight from the interviews stems from the rationale certain interviewees give for some of their rankings. From the given rationales we can conclude that professionals like to steer away from monoliths as much as possible. Their reasoning almost always includes bad experiences with monoliths in terms of working with them concerning scalability and deployment. This was observed mostly in the least and most experienced individuals. A reason for this could be that the less experienced individuals are steered more towards microservices from the start of their career due to market trends and the most experienced individuals have had more chance for bad experiences when monoliths were more common.

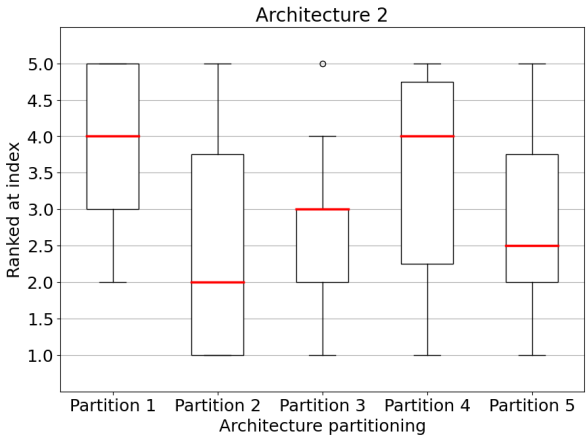
A second insight is the importance of additional metrics in the architectures themselves with respect to the edges in the graphs. The most important observed factor besides system context, is directional edges. Of the 10 interviewed individuals, 8 of them indicated that directional edges were important if not the most important factors dictating the quality of a partitioning. Things they watched for were bidirectional edges and cycles. The most important factor is bidirectional edges which frequently resulted in a partitioning that at least contains the bidirectional edge in one boundary. Cycles were also given as something to look out for with the added side note that good architecture almost never contains cycles to begin with as per the interviewees.

The rankings given by the interviewees are displayed in tables B.1, B.2, B.3, B.4 and B.5. In these tables, the rank of each partition given by each person is displayed. This means that each row represents an interviewee, each column represents a partition with the number corresponding to the number of the partition as displayed in section A and each cell represents the rank the interviewee gave that particular partition. With this, you can easily see how often each partition was ranked at what position.

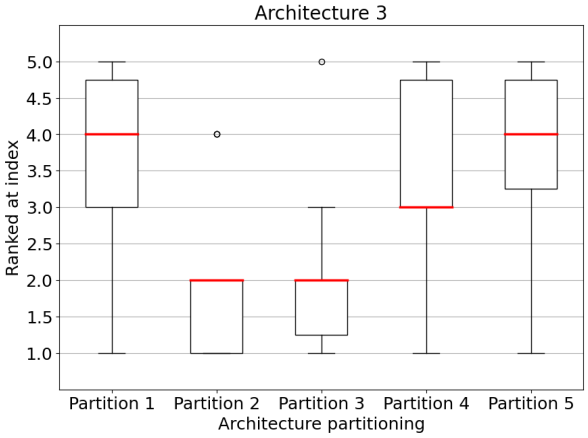
The first noticeable thing about the outcomes of the interviews is that they vary quite a bit across the different interviewees. However, this is to be expected as each person has different experiences and perceptions that influence their decisions. The most important insight, however, is the general consensus of the group of interviewees which gives insight into what is generally considered the best or highest quality architectural design. As explained in chapter 3, this can be visualised by plotting the data to box plots which filter outliers and give a median for each partition. These box plots are shown in figure 4.1.



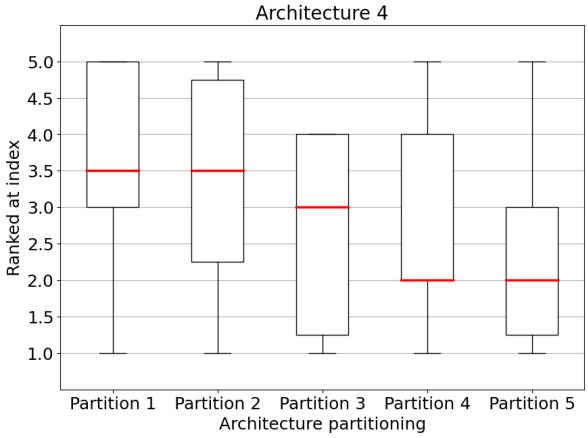
(a) Results for Architecture 1



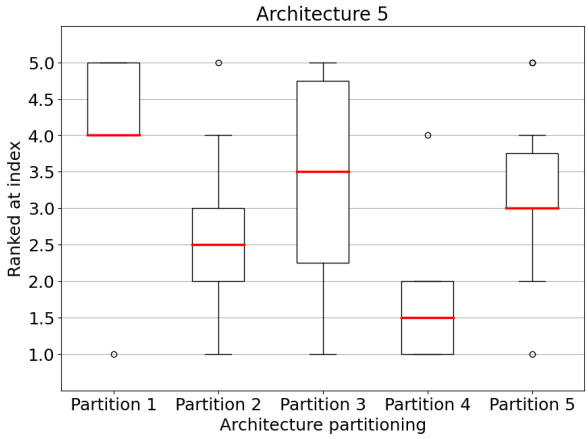
(b) Results for Architecture 2



(c) Results for Architecture 3



(d) Results for Architecture 4



(e) Results for Architecture 5

Figure 4.1: Plotted results of professional perceptions

These figures are to be interpreted as follows: The tails of the box plots give the maximum ranked index of that partition at the top and the minimum ranked index at the bottom respectively. This means that a partition was rated at a maximum or minimum on that respective index excluding outliers. The dots in the plots are outliers which differ too much from the general consensus which means they would unfairly skew the results if they are taken into account. The bottom and top of the boxes themselves are the 25^{th} and 75^{th} percentile respectively which means half of the participants chose between these edges and the other half outside of the box but within the maximum and minimum. The most important feature is the median as represented by the orange line situated somewhere within the box. This shows the median index of that partition which means the lower the median, the better the architecture. This

is because a lower index means closer to the number one spot which means the best partition.

The following insights can be drawn from these results as explained by the results from architecture 1 as shown in figure 4.1a: Looking at the medians of all partitions a ranking from best to worst partition can be made. This ranking would be partition 1, partition 2, partition 3, partition 5 and lastly, partition 4. This means that, in the perception of professionals, partition 1 is the qualitatively best partition and partition 4 is the worst for the given architecture. One more insight from this figure is that the interviewees are quite divided on partitions 1 and 4 as displayed by the large boxes and big tails. This can be contributed, as described earlier, to the previous experiences of the interviewees. This is also a consequence of such a highly connected diagram which is something you will rarely see as explained earlier. This contributes to the divided nature of the answers.

The ranking from best to worst as measured by experts for all architectures is summarized in table 4.1.

Architecture	Partition ranking				
	Best				Worst
Architecture 1	1	2	3	5	4
Architecture 2	2	5	3	4	1
Architecture 3	2	3	4	1	5
Architecture 4	5	4	3	2	1
Architecture 5	4	2	5	3	1

Table 4.1: Quality ranking of partitions for all measured architectures

In table 4.1, it is important to note some ranking decisions. These decisions involve the box plots which are almost the same. If two partitions have the same median but differ in box plot otherwise, the partition with the lowest maximum and minimum values or the lowest 25^{th} and 75^{th} percentiles is chosen as this still shows a slight preference. For example, this is the case in architecture 4 partitions 1 and 2. In this case, the medians are the same as well as the minimum and maximum values. However, the 25^{th} percentile of partition 2 lies slightly lower than the one of partition 1 which means there is still a very slight preference towards partition 2. No boxplots within these experiments are exactly the same.

With these results, it is also important to note that, in some cases, the professionals disagreed quite a bit about the optimal partition. This is shown by large boxes and big minimum and maximum values. This also means everything created or measured using professional perception will never be completely perfect as the professionals also never fully agree. Nevertheless, because the medians and boxplots still show preference in ranking, we are still able to draw conclusions about the general consensus.

4.2 Algorithm outcomes

In order to evaluate the ability of the different formulas to accurately measure the quality of an architecture, we first calculate the rankings these formulas give the different partitions. By then calculating the distance vector, we are able to calculate how far off these theoretical optimums are in comparison with the measured real-life optimums. The outcomes of the different formulas are displayed in tables 4.2, 4.3, 4.4, 4.5 and 4.6. If two outcomes have the same calculated complexity, both partitions are shown on both spots and split by a forward slash. This means these partitions are neither better nor worse than one or the other and are interchangeable. For each index in the ranking, the exact calculated complexity value is also given in the cell underneath said index. As stated before, if two indexes have the same complexity value they are separated by a forward slash but will have the same complexity meaning the exact complexity value is only given once for each cell. To shortly reiterate on the research method in chapter 3, the cohesion ranking will be from high to low as opposed to the other metrics since high cohesion is actually desirable. Another thing to take note of is the fact that a fully monolithic architecture does not have a coupling value as there are no components to have a coupling between. When normalizing the values this None value will be treated as a maximum along with the actual maximum value and will always get a value of 1.

Formula	Partition ranking and value Best				Worst
Perceived optimum	1	2	3	5	4
Industry formula	1	4	5	2	3
	258.39	886.96	1725.02	1876.55	1916.76
Coupling	2	5	3	4	1
	65.2	72.71	78.22	84.22	None
Cohesion	1	2	5	3	4
	1.0	0.57	0.31	0.25	0.0
Cluster	2	5	3	1/4	1/4
	72.0	77.0	78.0	90.0	90.0

Table 4.2: Quality ranking of partitions using formulas for architecture 1

Formula	Partition ranking and value Best				Worst
Perceived optimum	2	5	3	4	1
Industry formula	3	5	2	4	1
	159.51	168.18	207.30	234.45	630.35
Coupling	2	5	3	4	1
	15.52	19.23	23.53	83.40	None
Cohesion	1	2	3	5	4
	0.47	0.34	0.29	0.16	0.00
Cluster	2/3	2/3	5	1/4	1/4
	52.00	52.00	54.00	80.00	80.00

Table 4.3: Quality ranking of partitions using formulas for architecture 2

Formula	Partition ranking and value Best				Worst
Perceived optimum	2	3	4	1	5
Industry formula	4	5	2	3	1
	171.99	195.97	202.63	219.78	1258.93
Coupling	2	4	3	5	1
	7.28	34.93	50.44	115.77	None
Cohesion	1	2	3	4	5
	0.37	0.25	0.17	0.15	0.0
Cluster	2	3	4	1/5	1/5
	68.0	76.0	80.0	120.0	120.0

Table 4.4: Quality ranking of partitions using formulas for architecture 3

Formula	Partition ranking and value Best				Worst
Perceived optimum	5	4	3	2	1
Industry formula	2	3	4	5	1
	187.74	297.76	305.76	391.76	416.68
Coupling	3	2	5	4	1
	22.14	29.07	32.88	70.21	None
Cohesion	1	2	3	5	4
	0.56	0.28	0.09	0.04	0.00
Cluster	2	3	5	1/4	1/4
	50.00	52.00	60.00	70.00	70.00

Table 4.5: Quality ranking of partitions using formulas for architecture 4

Formula	Partition ranking and value Best				Worst
Perceived optimum	4	2	5	3	1
Industry formula	5	4	3	2	1
	3593.43	4165.45	4936.76	5023.92	24345.43
Coupling	2	3	4	5	1
	9.28	71.84	143.27	318.07	None
Cohesion	1	2	3	5	4
	0.14	0.05	0.04	0.03	0.02
Cluster	3	2	4	5	1
	637.0	648.0	652.0	729.0	988.0

Table 4.6: Quality ranking of partitions using formulas for architecture 5

One noticeable thing from these results is that the coupling measure and cluster measure are quite similar in most cases. With a fully connected architecture such as in table 4.2 the resulting ranking is almost exactly the same. With the other architectures, the differences are only a couple of indexes difference. This can be attributed to the fact the formulas for both these measures are similar but not exactly the same.

4.3 Reflecting accurate design rankings

In this section, we will look at how well the formulas perform in reflecting accurate design quality rankings. We do this because we are interested in how topology complexity is correlated with design quality. By looking at which formulas perform best in reflecting certain measurements and if they are accurate in their reflections, we will be able to gather insight into if and how the topology complexity is connected with quality.

4.3.1 Ranking comparison

In order to compare the initial rankings for each architecture, we will look at the measured optimal ranking as shown in table 4.1. We then compare this to the rankings given by the formulas as displayed in tables 4.2 to 4.6. This comparison is done on each individual index in a ranking of a formula to its relative position in the perceived expert optimum. The difference between the index of the formula ranking and the actual ranking is then added to a baseline of zero. This will result in some positive number which represents how far off the formula is from the measured optimal ranking. This means the closer to zero, the better the formula is at reflecting the professional opinion.

Lastly, since we are also interested in the sole optimum and top 3, we will calculate the same index difference for those measures. This means that again the closer to zero they are, the better. This will give a total insight into how accurate the formulas are in representing the total ranking, sole optimum and top 3.

Important to note is that if a formula has two partitions ranked the same, both rankings are tested in similarity on indexes and the eventual outcome is the average of the possibilities. The outcomes of these calculations are displayed in tables 4.7, 4.8, 4.9, 4.10 and 4.11.

Formula	Total difference	Optimum difference	Top 3 difference
Industry	8	0	4
Coupling	8	4	5
Cohesion	2	0	1
Clustering	7	3.5	4.5

Table 4.7: Ranking comparison for architecture 1

Formula	Total difference	Optimum difference	Top 3 difference
Industry	4	2	4
Coupling	0	0	0
Cohesion	8	1	3
Clustering	4	0,5	3

Table 4.8: Ranking comparison for architecture 2

Formula	Total difference	Optimum difference	Top 3 difference
Industry	10	2	6
Coupling	4	0	2
Cohesion	6	1	3
Clustering	1	0	0

Table 4.9: Ranking comparison for architecture 3

Formula	Total difference	Optimum difference	Top 3 difference
Industry	8	3	5
Coupling	8	2	6
Cohesion	12	3	6
Clustering	9	2	5.5

Table 4.10: Ranking comparison for architecture 4

Formula	Total difference	Optimum difference	Top 3 difference
Industry	6	1	5
Coupling	6	2	4
Cohesion	10	4	5
Clustering	6	2	3

Table 4.11: Ranking comparison for architecture 5

In order to draw some preliminary conclusions from these outcomes we take the averages of all

architectures in order to gain insight into the total performance of each formula for each measure. These averages are as follows:

Formula	Average total difference	Average optimum difference	Average top 3 difference
Industry	7.2	1.6	4.8
Coupling	5.2	1.6	3.4
Cohesion	7.6	1.8	3.6
Clustering	5.4	1.6	3.2

Table 4.12: Average ranking comparisons for all architectures

From the total results in table 4.12 we observe the following: The clustering and coupling formulas are the best when it comes to representing the complete measured optimal ranking as well as having the most accurate top 3. The distance of the top 3 is in line with the accuracy of representing the total opinion which is important as the best couple of partitions are often what is most important next to the sole optimum partition. For the sole optimum partitioning, we observe that all formulas except cohesion perform equally as well. This means we need to look at the results for the most accurate weights in combination with the ranking to conclusively tell which formula is the best for measuring the optimum partitioning.

This means our initial findings are that three of the four formulas are the most accurate at finding the sole optimal partition when it comes to assessing the design quality of a software architecture from the topology of system components. The clustering and coupling formulas seem to be the most accurate at both the complete ranking as well as the top 3.

4.3.2 Distance comparison

As described earlier, we will dive further into our mentioned findings about which formulas are best for the optimum partition and the cluster and coupling formulas are the best for the complete ranking. We do this by calculating the distance vector as described in chapter 3. The closer to 0 a distance value the better because this means the ranking is very similar. The normalized values of the formulas are easily calculated on account of them giving exact complexity values. However, humans do not give exact complexity values, meaning only normalizing the ranking would result in the assumption that the human ranking is perfectly uniformly distributed, which it is not. This is especially apparent in the earlier mentioned example of architecture 4 with partitions 1 and 2. In this case, the partitions are very close together with a preference towards partition 1. This means it is incorrect to assume uniform distribution as this would mean partitions 1 and 2 are equally distanced in comparison to all the other partitions which again, they are not. In order to mitigate this issue we calculate a value based on the index the partition is ranked at. With the most important factor being the median this becomes the starting value. The differences from the median to the 25^{th} and 75^{th} percentiles are then used to calculate whether the value needs to be skewed down or up slightly. Because these factors do not count nearly as much as the median in the ranking, these values are multiplied by a factor of 0.1. Important to note is that the factor of 0.1 is based on experimentation with outcomes and has no official research associated with it. A factor of one would be illogical since that would assume the percentiles weigh equally as heavy as the median and would result in different rankings than the result of only using the medians. In order to show slight preferences but not alter the original rankings, a factor of 0.1 was chosen. The influence of the percentiles will result in a value that is slightly below the median if the distance of the 25^{th} percentile to the median is larger than the distance of the median to the 75^{th} percentile or slightly above the median if opposite. If the distances of the 25^{th} and 75^{th} percentiles to the median are equal, the result will be the same as the exact median as this shows no stronger preference towards either side. The exact formula for calculating these values is as follows:

$$Value(P) = median(P) - 0.1(median(P) - 25^{th}(P)) + 0.1(75^{th}(P) - median(P))$$

With P being the partition and $25^{th}(P)$ and $75^{th}(P)$ being the values for the 25^{th} and 75^{th} percentile of partition P. For both the total average distance and the average optimum difference it counts that the closer to zero they are, the better the measurement. As will become apparent, the ability to find the optimum and the effectiveness of representing the total opinion do not always go hand in hand.

The normalized values of the complexity values for all measured architectures are given in tables 4.13, 4.14, 4.15, 4.16 and 4.17. Note that the values of the optimum, which represent the measured optimal partition, are now not uniformly distributed but alter in distance according to how close the boxplot values of the partitions are.

Formula	Normalized values Best				Worst
Optimum	0.000	0.151	0.519	0.623	1.000
Industry	0.000	0.379	0.884	0.976	1.000
Coupling	0.000	0.395	0.685	1.000	1.000
Cohesion	0.000	0.430	0.690	0.750	1.000
Cluster	0.000	0.278	0.333	1.000	1.000

Table 4.13: Normalized complexity values for architecture 1

Formula	Normalized values Best				Worst
Optimum	0.000	0.260	0.429	0.948	1.000
Industry	0.000	0.018	0.101	0.159	1.000
Coupling	0.000	0.055	0.118	1.000	1.000
Cohesion	0.000	0.277	0.383	0.660	1.000
Cluster	0.000	0.000	0.071	1.000	1.000

Table 4.14: Normalized complexity values for architecture 2

Formula	Normalized values Best				Worst
Optimum	0.000	0.012	0.607	0.988	1.000
Industry	0.000	0.022	0.028	0.044	1.000
Coupling	0.000	0.255	0.398	1.000	1.000
Cohesion	0.000	0.324	0.541	0.595	1.000
Cluster	0.000	0.154	0.231	1.000	1.000

Table 4.15: Normalized complexity values for architecture 3

Formula	Normalized values Best				Worst
Optimum	0.000	0.111	0.571	0.937	1.000
Industry	0.000	0.481	0.516	0.891	1.000
Coupling	0.000	0.144	0.223	1.000	1.000
Cohesion	0.000	0.500	0.839	0.929	1.000
Cluster	0.000	0.100	0.500	1.000	1.000

Table 4.16: Normalized complexity values for architecture 4

Formula	Normalized values				
	Best				Worst
Optimum	0.000	0.385	0.606	0.769	1.000
Industry	0.000	0.028	0.065	0.069	1.000
Coupling	0.000	0.203	0.434	1.000	1.000
Cohesion	0.000	0.750	0.833	0.917	1.000
Cluster	0.000	0.031	0.043	0.262	1.000

Table 4.17: Normalized complexity values for architecture 5

The distance results, which are a direct correlation to the performance of the formulas, can be found in tables 4.18, 4.19, 4.20, 4.21 and 4.22. In these tables, the values are the individual distances meaning the difference between normalized ranking values squared. This means that a value of 0 is given when a partition is ranked at the correct spot with the correct weight. This in turn allows us to calculate the total distance vector by taking the square root of the sum of all the individual distances as described in chapter 3. These total distances are given in the two last rows which are ultimately the most important as they give the total score for the respective method in the ability to assess architecture quality as measured against professional opinion for that architecture. Lastly, the distance measure for the professional opinion or "optimum" is excluded since you would compare the measure against itself which is redundant.

Important to note is that in the case that two rankings of a formula are the same, a distinction between outcomes is not needed since they share the same complexity value and will therefore result in the same normalized value.

Formula	Industry	Coupling	Cohesion	Cluster
	0.000	0.023	0.000	0.023
	0.386	0.052	0.078	0.119
	0.068	0.027	0.005	0.034
	0.680	0.000	0.053	0.000
	0.231	1.000	0.000	1.000
Total distance	1.169	1.050	0.369	1.084
Optimum distance	0.000	1.000	0.000	1.000

Table 4.18: Distance vectors of formulas for architecture 1

Formula	Industry	Coupling	Cohesion	Cluster
	0.184	0.000	1.000	0.184
	0.058	0.042	0.077	0.000
	0.010	0.096	0.002	0.035
	0.622	0.003	0.160	0.003
	0.000	0.000	0.003	0.000
Total distance	0.935	0.376	1.114	0.471
Optimum distance	0.101	0.000	0.277	0.000

Table 4.19: Distance vectors of formulas for architecture 2

Formula	Industry	Coupling	Cohesion	Cluster
	0.369	0.000	0.976	0.000
	0.956	0.124	0.105	0.020
	0.001	0.149	0.279	0.142
	0.001	0.000	0.000	0.000
	0.000	0.000	0.000	0.000
Total distance	1.152	0.523	1.167	0.402
Optimum distance	0.028	0.000	0.324	0.000

Table 4.20: Distance vectors of formulas for architecture 3

Formula	Industry	Coupling	Cohesion	Cluster
	0.877	0.327	1.000	0.877
	0.008	0.628	0.191	0.222
	0.164	0.050	0.072	0.250
	0.794	0.790	0.862	0.790
	0.000	0.000	0.790	0.000
Total distance	1.358	1.340	1.707	1.463
Optimum distance	0.891	0.223	0.929	0.500

Table 4.21: Distance vectors of formulas for architecture 4

Formula	Industry	Coupling	Cohesion	Cluster
	0.367	0.148	1.000	0.592
	0.001	0.321	0.134	0.125
	0.496	0.188	0.004	0.002
	0.100	0.155	0.097	0.118
	0.000	0.000	1.000	0.000
Total distance	0.982	0.902	1.495	0.915
Optimum distance	0.028	0.434	1.000	0.043

Table 4.22: Distance vectors of formulas for architecture 5

The first architecture in table 4.18 is the perfect example of a formula not representing the complete opinion the best but still having the ability to select the best optimum. For the architecture in table 4.18 the industry formula does not represent the total opinion very well with it having a total distance of 1.169 in comparison to the measured baseline in table 4.1. Apart from the optimum partition 1, the formula has all other partitions at different spots. This results in a low representation of the overall ranking but a good representation of the optimum. However, in the same case, we can see the cohesion measurement is very accurate in aligning with the actual ranking as well as giving the correct optimum.

With these results, we can create an average number for each formula on how accurate they represent the overall opinion and how close they get to getting the right optimum on average. With this result, the same idea applies that the closer to zero the outcome, the better. This gives the following result:

Method	Average total distance	Average optimum distance
Industry formula	1.119	0.210
Coupling	0.838	0.331
Cohesion	1.170	0.506
Cluster	0.867	0.309

Table 4.23: Total average results for all tested formulas

The outcomes in table 4.23 are in-line with our original finding that the clustering and coupling formulas are best at the total ranking. What can be observed is that the coupling formula performs slightly better using the distance vector. However, the clustering formula has a considerably better average distance to the optimum partition which means the cluster formula can still be considered better.

From the results, we can also clearly observe that the industry formula is the most accurate at finding the optimum partitioning with the most accurate weight. This means we conclude that this formula is best used to find the optimum partitioning when measuring the complexity of a topology. From the insight that the industry formula and clustering formula are the best and the fact the distance vectors are fairly low, we can conclude that topology complexity is a good method for showing design quality. With the formula from industry practice, we are able to accurately find the optimum partitioning from a list of possible partitions with an accurate weight in comparison to the other partitions in said list. The Clustering formula is an accurate complexity measurement to rank a complete set of partitions with accurate distance within said ranking.

4.3.3 Context influence

As stated in chapter 3, we also tested the influence of context on the answers of the professionals. Given architecture 3 with context as shown in section A.6, the experts chose an entirely different partition as opposed to the ones shown in section A.4 despite being given the chance to elect a partition created by themselves for the number one spot. The most chosen partition given this context is as follows:

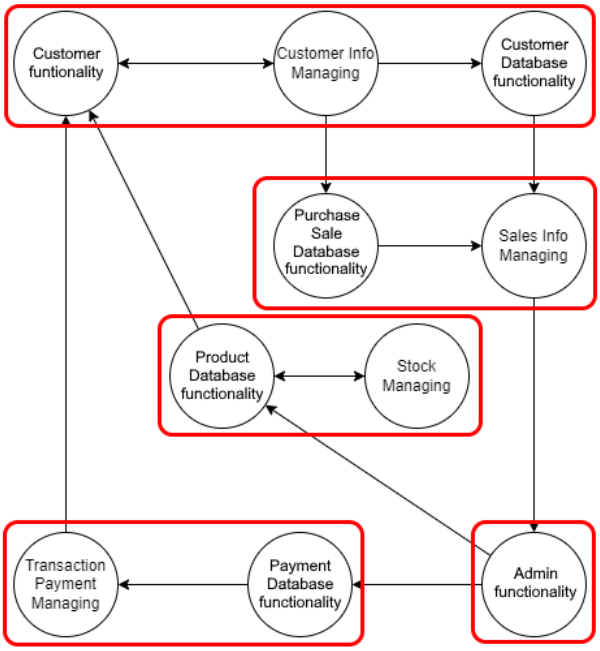


Figure 4.2: Best quality partition as measured by expert opinion

This result shows us that context matters a lot for the eventual outcome. A formula unfortunately cannot process context very well meaning we have to rely on the learnings in order to better the formula. The most important observation is that the earlier mentioned insight of bidirectional edges is still upheld

given the context. However, in the architecture without context, one bidirectional edge was broken for the sake of containing the square nodes within one boundary. This is something that can be taken into account when utilizing the industry formula by rarely breaking bidirectional edges in the inputs or looking for these square formations with one edge going in and one edge going out. More often than not the architect using the industry formula will not input possibilities where the bidirectional edges are broken meaning that does not have to be a problem.

What becomes especially apparent about the industry formula is the fact that it steers towards smaller partitions. From the results, we can see that the formula often chooses boundaries with one, two or three aggregates. This can be seen from the rankings of this formula for the fifth architecture shown in section A.5. The rankings, as displayed in table 4.6, go from partitioning 5 to 1 in numerical order. If we look at the partitions in section A.5, it is clear that the partitions are ordered from 1 to 5 in granularity ranging from fully monolithic in partition 1 to split up the smallest in partition 5. From this, we can conclude that the industry formula steers towards smaller partitions which is not always the best approach according to professional opinion. This steering can be attributed to the fact that partitions only containing one node have a complexity value of 1. This is far less than all other numbers which increase exponentially by a factor of 3.1 as can be seen in the formula in chapter 3.

4.4 Assessing architecture quality using complexity

Our outcomes show that not one formula is the sole best at assessing the quality of the architecture design but rather is divided amongst two formulas. The overall results as shown in table 4.23 show that the industry formula is generally the best at finding the best architecture measured in terms of quality. The coupling and cluster formulas give the most accurate representation of the professional opinion overall. The cluster formula does however give a more accurate representation of the optimum in comparison to the coupling formula. This means these formulas would be the best options to assess the quality of an architecture using complexity as an indicator. Which formula is best used depends on what is needed exactly. As described in chapter 1, more often than not the architect would like to know the highest quality architecture given a range of partitions. In that case, the industry formula is best used to give insight into what option would be best since this formula gives the most accurate representation of the optimal partition. If the architect is interested in a ranking given a range of options, the coupling and cluster algorithms would be the best in giving the best approximation of the right total ranking. Due to the fact that the clustering formula gives a better representation of the optimum, this would be the better method. This means it highly depends on the situation whether the industry formula or the clustering formula should be used. If the project is at the very start and the system is being built from the ground up or a fully monolithic system is being migrated to micro-serviced, it might be best to create a large range of different partitions and use the coupling and clustering algorithms to get a general feel for the overall ranking of these architectures. This allows the architect to quickly visualize all the options and at what end of the ranking the options lie. The industry formula could then be used to get insight into what the sole optimum architecture is and whether that is in line with the earlier created ranking. The industry formula might best be used when the project is up and running and small alterations to the architecture need to be made. The architect could create two or three different options in this case and use the industry formula to quickly get insight into whether their complexity numbers vary greatly or not. In the case that a small alteration results in a great difference in complexity the architect can use this to easily evaluate whether this change is really for the good or not.

Chapter 5

Discussion

In this chapter, we discuss the results of our tests of the different formulas in their ability to measure the quality of architectures.

5.1 Findings

From the experiments, we can formulate a couple of findings relating to using topology complexity as a measure of the quality of architectures.

> **Finding 1:** The formula from industry practice is the best of the measured formulas at indicating the optimal architecture design according to measured professional opinion.

Finding 1 follows from the average distance to the optimum. The average amount of indexes the calculated optimum of this formula differed from the measured real-life optimums was an average of 1.6 index places difference. This is similar to other formulas. The finding follows from the average distance vector to the optimum of the calculated complexity values which lies at 0.210. This distance to the optimum shows that the industry formula performs considerably better than the other formulas as compared to the second-lowest distance of 0.309. Despite this, the accuracy in correctly representing the total overall opinion of this formula performed not the best compared to the other formulas. As mentioned earlier, this can be attributed to the fact the new formula tends to steer towards very few aggregates within one boundary. This is not necessarily bad since we are creating microservices but it most often is not the best solution for a complete ranking either.

> **Finding 2:** From the tested formulas, the coupling and clustering complexity measures are the best methods to indicate the quality of an architecture design for a complete ranking of partitions.

As stated in finding 2, the tested coupling and clustering formulas were best at representing the complete measured ranking of all partitions. With an average distance of 0.838 and 0.867 respectively across the five different architectures, these formulas performed considerably better than the other formulas. The clustering formula gives a more accurate number for the optimal partition in comparison to the coupling formula with an average optimum distance of 0.309 compared to 0.331 for the coupling formula. This means the clustering formula can be considered better as the difference between the optimum distances is slightly more important than the distance between the average total distances. This also means the clustering formula is best used in combination with the industry formula mentioned in finding 1.

> **Finding 3:** Directionality in a graph is important for creating partitions. Bidirectional edges most often go together as well as square or triangular formations with one entry and one exit edge.

Finding 3 follows directly from the interviews. The majority of the interviewees indicated that bidirectional arrows almost always go together especially given some context from which it is made clear why said edge is bidirectional. Another thing to look out for is simple structures such as squares or

triangles that have one entry point and one exit point. The interviewees indicated that these structures were very easy to convert into microservices since they have very few dependencies to control with only one incoming and outgoing edge.

5.2 Used method

In this research, we used the opinion of professionals in the field to gauge the quality of various architectures. We did this by interviewing individuals with enough experience so that we could create a baseline against which we could measure various formulas. By using this method, we were able to connect the theoretical highest-quality architectures with the real-world perceived highest-quality architectures. By using the insights of professionals we believe we got a good measurement on what architecture partitions were of a good quality and which were not. By comparing the rankings of the formulas against the baseline, we were able to show that at least two of these formulas were a good indicator of the quality of architectures.

We believe this method could be further improved using measurements from an actual implemented system. This would allow for our findings about architectures to be translated into working models which is the next step in developing microservice systems. By implementing a real system using the optimal formulas from findings 1 and 2, further insight can be created if the quality of the architecture also translates into a good-performing system.

5.3 Threats to validity

The main threat to the validity of this research stems from the limited scope the professional opinion sets. Because the interviews were conducted with experts within the research company, the argument can be made that these persons are affected by the way of working within the company. Having said that, the interviews showed different insights with sometimes very different opinions between participants.

Another pitfall of this research is the type of systems this research can be applied to. Because the experts have different work experiences with different systems and clients, there is not one system for which this research is fully optimized. A more in-depth analysis with experts working on very similar systems between them would be needed to conclusively say that the findings work on that specific type of system.

Lastly, it is important to show the assumptions made in this research as they can have an effect on the eventual outcome. One assumption made within this thesis is the fact that each professional opinion of the interviews is weighed equally to the opinions of other interviewees. The argument can be made that the more experienced an individual is, the more that person's opinion should count. Similarly, the argument can be made that a more experienced individual has had more time to learn the wrong things and more importantly, has had more possibilities to form unfair opinions against certain architectures. This became apparent during the interviews during which some individuals stated they purposely steered away from monoliths as the optimal solution due to bad experiences. It is due to this reasoning and the fact both arguments have a strong basis that we decided to treat every professional opinion equally despite years of experience. The only important factor we took into account was the fact that an interviewee did need at least 5 years of experience with such systems in order to be able to argue why their opinion was valid within this research.

Chapter 6

Conclusion

In this thesis, we answer the main research question of *"How to effectively assess the design quality of software architecture from the topology of system components?"*. To answer this question we researched three additional sub-questions which help substantiate our conclusion to the main research question.

SubQ 1: *What are existing approaches for measuring architecture topology complexity?*

We researched what current methods exist for measuring architecture complexity. From related work, we concluded that various methods exist with different useability. The state of the art methods greatly relies on the usage of weighted edges to create partitions for a possible microservice architecture. Our research looked at situations where weighed edges are not always known due to limited information or the stage at which a project currently resides. Three different measures have been found that measure the complexity of an architecture given a certain partitioning and thus not requiring weighted edges. These three measures are a coupling metric, a cohesion metric and a clustering metric. Our research adds another formula to this list which has not yet been tested for its accuracy. This new formula has been adapted from an older also not tested formula within the research company Info Support. These four methods make up the existing approaches for measuring architecture topology complexity without having the need for edge weight.

SubQ 2: *How is topology complexity correlated with the design quality?*

Following this, we researched the accuracy of the different formulas to represent measured optimal architecture designs using complexity as a metric. We used the opinion of professionals in the field with 5 to 20 years of experience to create a baseline to compare the formulas. After creating a baseline measure for architecture quality, we calculated the optimum and complete ranking according to each formula. After comparing the formula outcomes against the real-life perceived optimums, we concluded that the industry formula performs best in terms of calculating the optimal partition. The clustering metric is best used to give the most accurate representation of the complete baseline ranking overall. This also means the usability of these two formulas differs per use case. In this research, we have shown that these two formulas perform well in representing professional opinion. This means that the topology complexity that is calculated using said formulas and is used for creating the ranking of possible topologies is a good measurement to be used to assess the design quality of an architecture. Even though design quality relies on more than only complexity, we believe this research shows there is a good correlation between these two aspects.

SubQ 3: *How to assess the architecture quality using the indicator of complexity?*

How to assess the architecture quality using a formula that calculates complexity depends on the context the formula is used in. As shown in chapter 3, the industry formula performs best in representing the optimal architecture partition with the average amount of index places the calculated optimum differs from the actual measured optimum being 1.6. This means this formula is useful when using it in a context where one optimum needs to be chosen or when an architect wants to know if a small alteration to the architecture makes a large difference in the optimum partitioning. While the industry formula is best at calculating the sole optimum, the clustering formula is the most accurate at representing the total ranking of partitions. The coupling formula performs equally in representing the total ranking of partitions but performs worse in finding the optimal partition compared to the clustering formula. This means the

clustering formula is best used when an architect wants to gain a broader understanding of the general ranking of a set of created partitions as the optimum partition in itself is very important. With this method, an architect is able to get a feel for what partitions are generally the best quality and what partitions are the worst quality. This means the situation highly dictates how to asses the architecture quality using the indicator of complexity. By following our findings in using either the industry formula or the clustering formula, a good insight can be gathered into the quality of the architecture topology.

> ***RQ:*** *How to effectively assess the design quality of software architecture from the topology of system components?*

The measuring of the complexity of an application using the formulas in this thesis focuses on architectures that use aggregates and program modules as nodes and dependencies and calls as edges. Partitions are then created based on the number of nodes, the number of edges crossing partition boundaries and the number of edges within a partition. By creating a baseline measurement of what is perceived to be a high-quality architecture by professionals in the field, we were able to measure and compare the accuracy of the formulas in accurately measuring the quality of designs. From these interviews, we also gained insight into what to look for in a high-quality partitioning which help answer the main research question.

Therefore, the main research question of this thesis is answered by taking the context into account. In this thesis, we have shown that complexity is a good indicator of the design quality of software architecture using the topology of system components. In the case that architecture specifics such as edge weights are not known yet, the newly tested industry formula in this thesis and the clustering algorithm as described in [16] are the best candidates. The industry formula can effectively be used to gain a general understanding of the optimal partition. The clustering formula can be used to gain the most accurate understanding of a ranking of a list of possible partitions to not only assess which partitions are of low complexity and thus high quality but also which ones are of high complexity and thus low quality. Another method of effectively measuring the complexity of an application is by creating small alterations to the original architecture partitioning and comparing the calculated complexity numbers. Since both the industry formula and cluster formula produce larger numbers, a small difference in the output complexity of the formulas means both topologies can be considered viable options. If the complexity number changes greatly between two partitions using these formulas, the architect will quickly and effectively know what design is of the higher quality.

Chapter 7

Future work

7.1 Formula recommendation

There are a couple of factors that can improve the new formula which would need further validation. In this chapter, we will focus on explaining what could be altered or added to the perceived best formula to create an even better formula for measuring the quality of architecture.

From the interviews, we concluded that direction is very important when creating partitions. In particular, bidirectional edges are perceived to be very important and should not be broken by partitions. This alteration can be made by additionally increasing the calculated complexity number for every bidirectional edge that is broken. Another point gathered from the interviews was that a differentiation can be made between incoming and outgoing edges depending on the type of edge. If the edge represents an asynchronous callback, the outgoing edge could weigh less than when the data is received back since the service can go on running other tasks. This differentiation could allow the formula to create more specific partitions which might help with the overall usefulness.

From the related work in chapter 2, we concluded there already exist several algorithms that create partitions using weights. During the interviews, the interviewees also mentioned that edge weights always help with creating partitions and evaluating the quality of a certain partitioning. However, as mentioned in chapter 1, these system specifics are not always known at the time of system creation which would mean the architect would need to guess the weights which can steer towards an unwanted solution due to wrong weights. One possibility would be to assess how often edges are taken using a proof of concept system and create weights based on that. The original formula does have the ability to create stronger bonds, thus emulating higher-weighed edges, by drawing one edge as multiple ones which increases the negative effects of breaking said edge.

7.2 Method extension

As mentioned in chapter 5, we would like to further our findings by taking the formulas into practice. By using our findings of the two best formulas to create an actual system, we would be able to gather information about the performance of a system. This would allow for the comparison in performance which is another good metric for the evaluation of architectures. By comparing the outcomes of the formulas to the outcomes of the performance tests, we would be able to show if the conclusions also translate to performance metrics as opposed to only architecture quality.

7.3 Interview group range

Another useful future addition to this research would be to interview more people from different work cases. This could result in more specific results where the formula might perform better for certain systems than for others. This would add to the usefulness of the research because it would allow for a more focused argumentation on why you should or should not use a certain formula in a certain situation.

Acknowledgements

I want to thank my university supervisor Zhiming Zhao for helping me create this thesis. For the same reason and continued support throughout this thesis, I also want to thank Jan-Jelle Kester, my company supervisor. Lastly, I would also like to extend my gratitude towards Info Support for providing the environment to conduct this research.

Bibliography

[1] S. Baškarada, V. Nguyen, and A. Koronios, "Architecting microservices: Practical opportunities and challenges," *Journal of Computer Information Systems*, vol. 60, no. 5, pp. 428–436, 2020.

[2] M. Fowler and J. Lewis, *Microservices, a definition of this new architectural term*, Mar. 2014. [Online]. Available: `https://martinfowler.com/articles/microservices.html`.

[3] T. Cerny, M. J. Donahoo, and M. Trnka, "Contextual understanding of microservice architecture: Current and future directions," *ACM SIGAPP Applied Computing Review*, vol. 17, no. 4, pp. 29–45, 2018.

[4] A. Bucchiarone, N. Dragoni, S. Dustdar, S. T. Larsen, and M. Mazzara, "From monolithic to microservices: An experience report from the banking domain," *Ieee Software*, vol. 35, no. 3, pp. 50–55, 2018.

[5] H. Vural and M. Koyuncu, "Does domain-driven design lead to finding the optimal modularity of a microservice?" *IEEE Access*, vol. 9, pp. 32 721–32 733, 2021. DOI: `10.1109/ACCESS.2021.3060895`.

[6] K. Srinivasan and T. Devi, "Software metrics validation methodologies in software engineering," *International Journal of Software Engineering & Applications*, vol. 5, no. 6, p. 87, 2014.

[7] E. J. Weyuker, "Evaluating software complexity measures," *IEEE transactions on Software Engineering*, vol. 14, no. 9, pp. 1357–1365, 1988.

[8] R. Session. "The mathematics of it simplification." (Apr. 2011), [Online]. Available: `http://www.rogersessions.com/library/white-papers#the-mathematics-of-it-simplification`.

[9] M. Villamizar *et al.*, "Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud," in *2015 10th Computing Colombian Conference (10CCC)*, IEEE, 2015, pp. 583–590.

[10] N. Dmitry and S.-S. Manfred, "On micro-services architecture," *International Journal of Open Information Technologies*, vol. 2, no. 9, pp. 24–27, 2014.

[11] Z. Li, C. Shang, J. Wu, and Y. Li, "Microservice extraction based on knowledge graph from monolithic applications," *Information and Software Technology*, vol. 150, p. 106 992, 2022.

[12] G. Mazlami, J. Cito, and P. Leitner, "Extraction of microservices from monolithic software architectures," in *2017 IEEE International Conference on Web Services (ICWS)*, IEEE, 2017, pp. 524–531.

[13] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of statistical mechanics: theory and experiment*, vol. 2008, no. 10, P10008, 2008.

[14] N. Santos and A. R. Silva, "A complexity metric for microservices architecture migration," in *2020 IEEE international conference on software architecture (ICSA)*, IEEE, 2020, pp. 169–178.

[15] E. Allen, "Measuring graph abstractions of software: An information-theory approach," eng, in *Proceedings Eighth IEEE Symposium on Software Metrics*, IEEE, 2002, pp. 182–193, ISBN: 0769513395.

[16] L. A. Belady and C. J. Evangelisti, "System partitioning and its measure," *Journal of Systems and Software*, vol. 2, no. 1, pp. 23–29, 1981.

[17] M. Shepperd, "A critique of cyclomatic complexity as a software metric," *Software Engineering Journal*, vol. 3, no. 2, pp. 30–36, 1988.

[18] Y. Ma, K. He, and D. Du, "A qualitative method for measuring the structural complexity of software systems based on complex networks," in *12th Asia-Pacific Software Engineering Conference (APSEC'05)*, IEEE, 2005, 7–pp.

[19] M. G. Patulada, *Dfd for inventory management system*, Sep. 2022. [Online]. Available: https://itsourcecode.com/uml/dfd-for-inventory-management-system/?expand_article=1.

Acronyms

AWS Amazon Web Services. 9
DDD Domain Driven Design. 4, 6, 9, 11, 13, 17, 18
UVA University of Amsterdam. 8

Appendix A

Tested architectures

A.1 Architecture 1

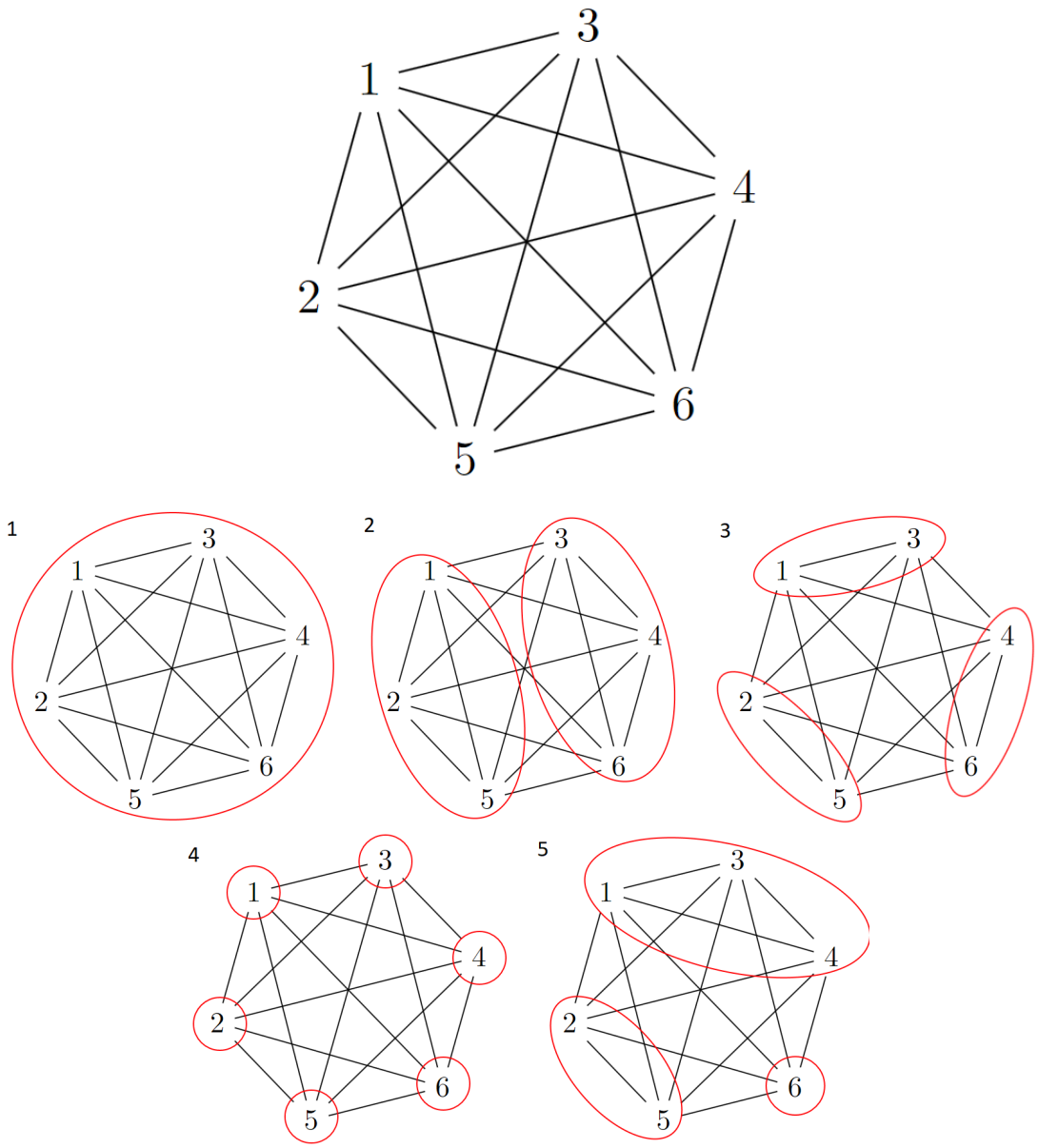


Figure A.1: Architecture 1 and the presented partitions

A.2 Architecture 2

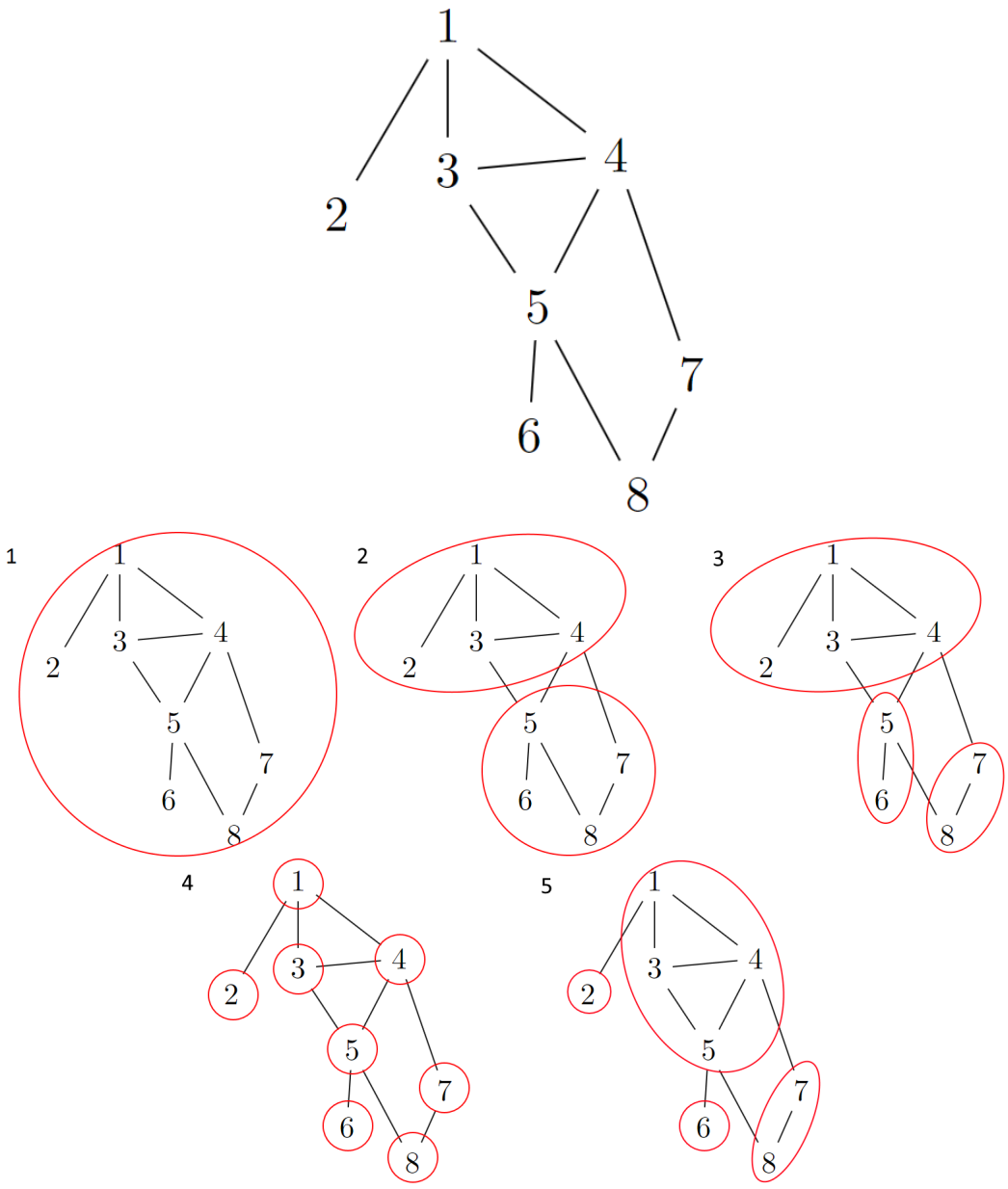


Figure A.2: Architecture 2 and the presented partitions

A.3 Architecture 3

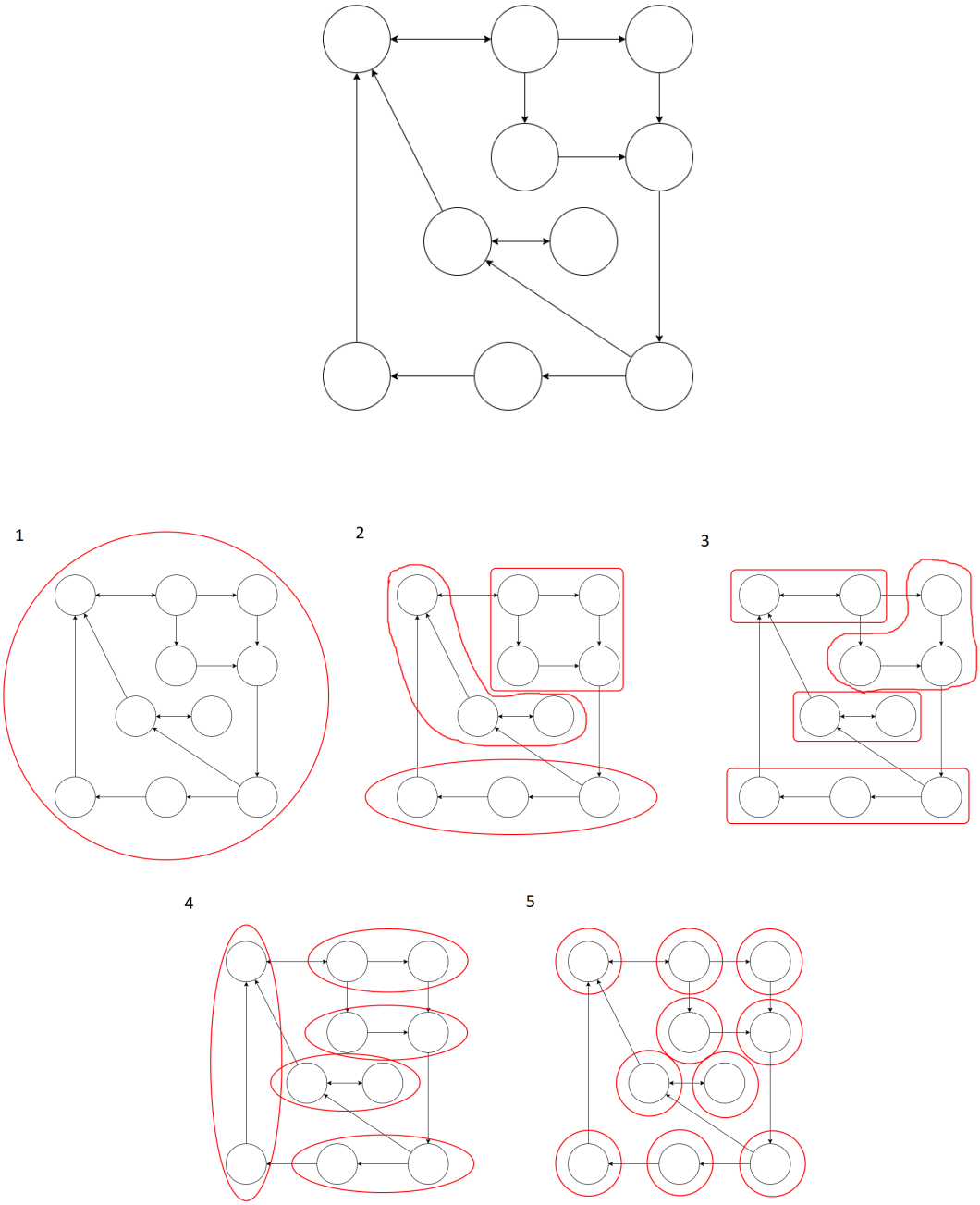


Figure A.3: Architecture 3 and the presented partitions

A.4 Architecture 4

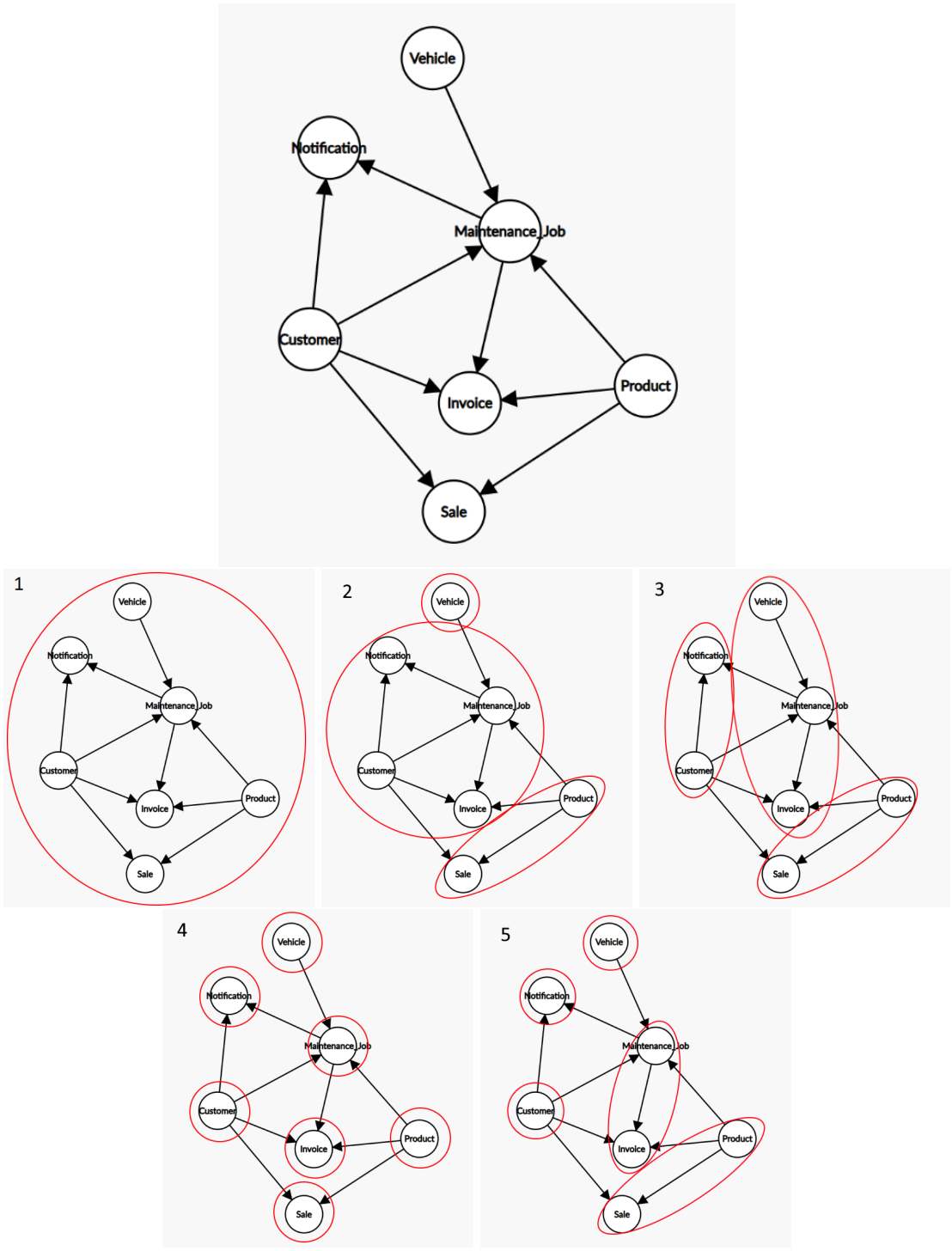
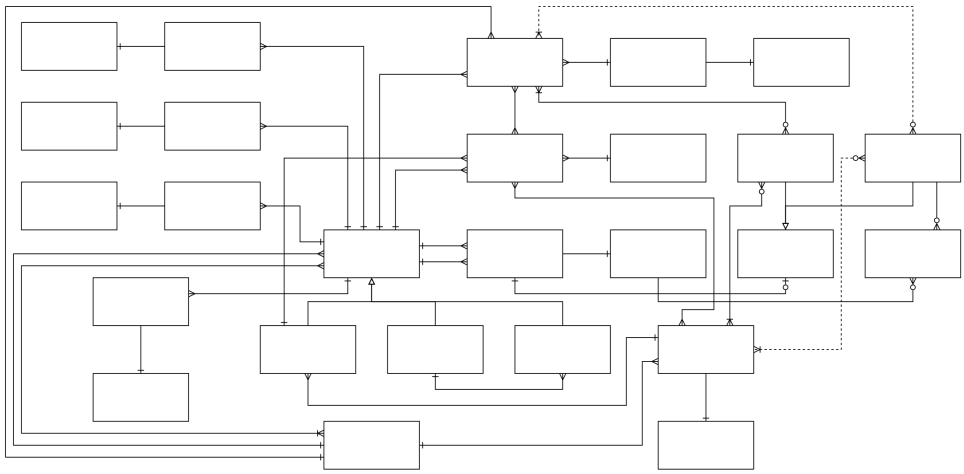
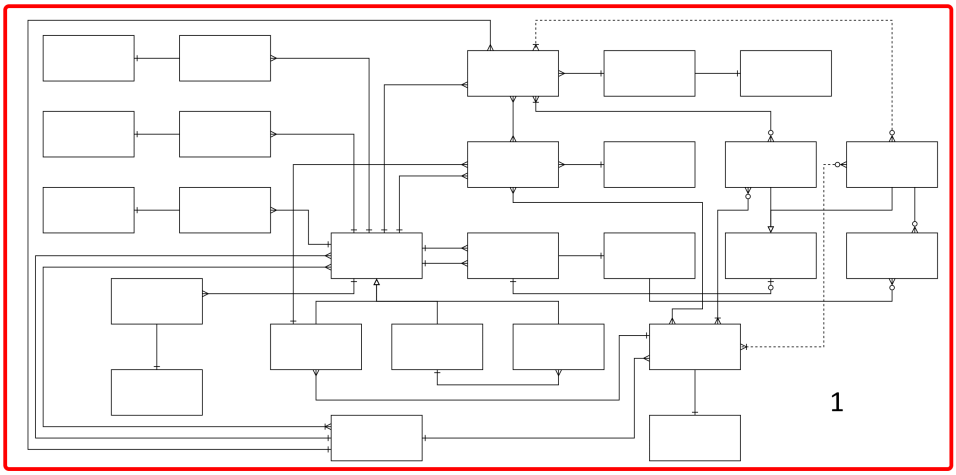
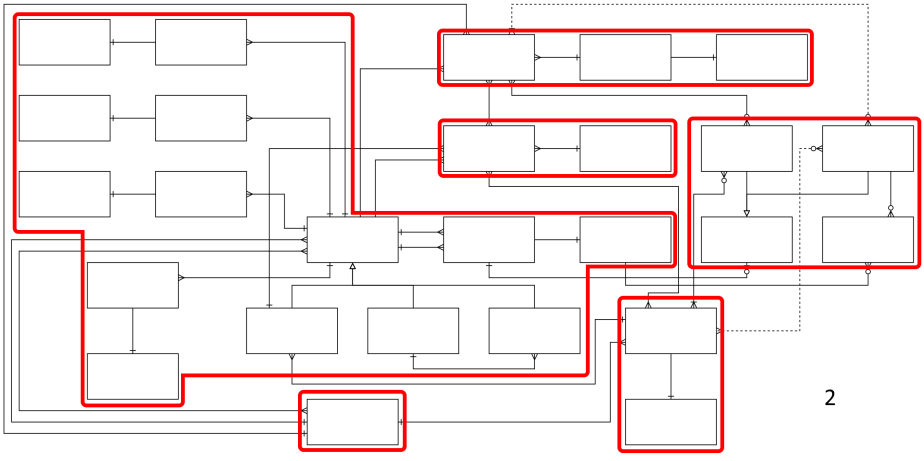


Figure A.4: Architecture 4 and the presented partitions

A.5 Architecture 5







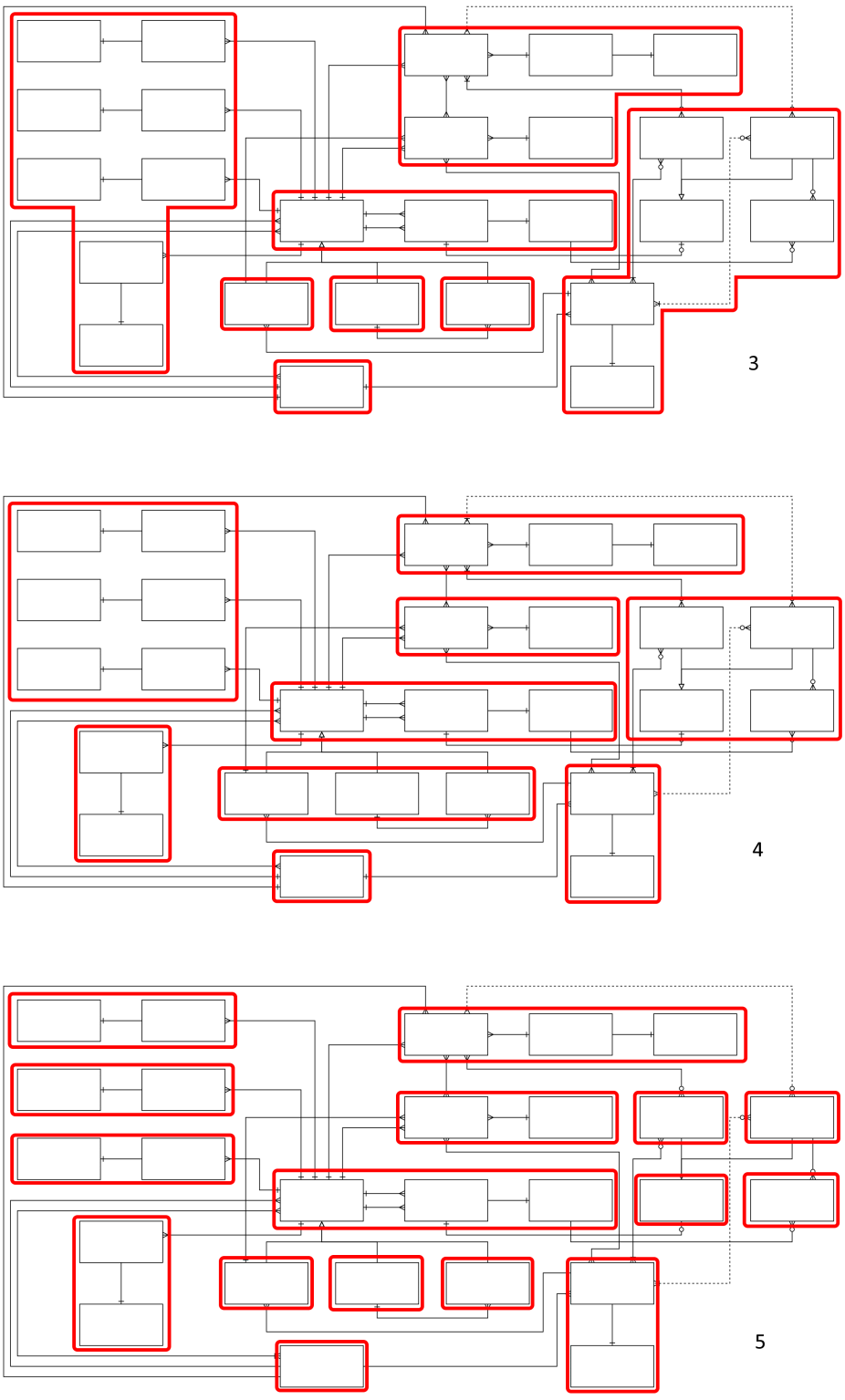


Figure A.5: Architecture 5 and the presented partitions

A.6 Architecture 6

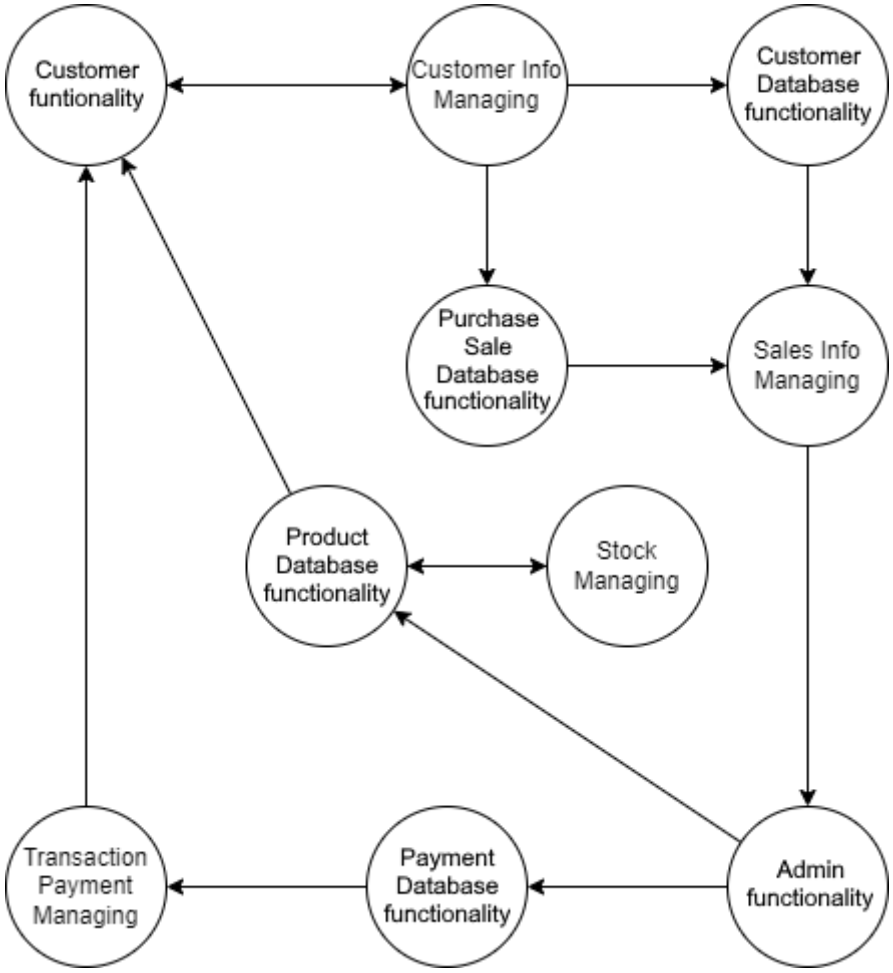


Figure A.6: Architecture 3 with context, referred to as architecture 6 [19]

Appendix B

Professional perception outcomes

Interviewee	Partition n ranked at index				
	1	2	3	4	5
Person 1	1	2	3	5	4
Person 2	1	2	3	5	4
Person 3	1	2	4	5	3
Person 4	1	3	4	2	5
Person 5	1	3	4	2	5
Person 6	4	2	3	5	1
Person 7	4	2	3	5	1
Person 8	5	3	2	4	1
Person 9	5	1	2	4	3
Person 10	2	5	3	1	4

Table B.1: Rankings of architecture 1

Interviewee	Partition n ranked at index				
	1	2	3	4	5
Person 1	5	4	2	3	1
Person 2	2	5	3	1	4
Person 3	4	1	3	5	2
Person 4	3	4	5	2	1
Person 5	5	1	3	4	2
Person 6	2	1	3	5	4
Person 7	4	2	1	5	3
Person 8	5	3	1	4	2
Person 9	5	1	2	4	3
Person 10	3	2	4	1	5

Table B.2: Rankings of architecture 2

Interviewee	Partition n ranked at index				
	1	2	3	4	5
Person 1	5	2	1	3	4
Person 2	3	4	2	5	1
Person 3	4	2	1	3	5
Person 4	3	2	1	5	4
Person 5	1	4	5	2	3
Person 6	3	1	2	4	5
Person 7	4	1	2	3	5
Person 8	5	2	3	1	4
Person 9	5	1	2	3	4
Person 10	4	1	2	5	3

Table B.3: Rankings of architecture 3

Interviewee	Partition n ranked at index				
	1	2	3	4	5
Person 1	5	4	3	1	2
Person 2	3	5	4	1	2
Person 3	3	5	1	4	2
Person 4	3	5	4	2	1
Person 5	1	3	4	2	5
Person 6	4	3	2	5	1
Person 7	3	2	1	5	4
Person 8	5	4	3	2	1
Person 9	5	2	1	4	3
Person 10	5	1	4	2	3

Table B.4: Rankings of architecture 4

Interviewee	Partition n ranked at index				
	1	2	3	4	5
Person 1	4	3	5	1	2
Person 2	4	5	3	2	1
Person 3	5	4	1	2	3
Person 4	4	1	5	2	3
Person 5	5	3	1	2	4
Person 6	4	3	2	1	5
Person 7	1	2	5	4	3
Person 8	5	2	4	1	3
Person 9	4	2	3	1	5
Person 10	5	2	4	1	3

Table B.5: Rankings of architecture 5