IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received January 15, 2021, accepted February 5, 2021, date of publication February 22, 2021, date of current version March 2, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3060895

Does Domain-Driven Design Lead to Finding the Optimal Modularity of a Microservice?

HULYA VURAL[1], **(Member, IEEE), AND MURAT KOYUNCU**[2]
[1]Information & Communication Technology Department, Havelsan Inc., 06510 Ankara, Turkey
[2]Department of Information Systems Engineering, Atilim University, 06830 Ankara, Turkey

Corresponding author: Hulya Vural (hulya.vural.tr@gmail.com)

ABSTRACT Information systems are moving into the cloud. The new requirements enforced by cloud standards are high availability, high scalability, and a reduced mean time to recovery. Due to these new requirements, information system architecture styles are also evolving. Microservice architecture is becoming the de facto standard for developing highly modular cloud information systems. Since microservices were introduced, there has been an ongoing debate concerning how to choose the granularity of a microservice. In this study, the optimal point of granularity for microservices is examined based on coupling and cohesion values. The present study is based on two design examples generated in previous studies that applied domain-driven design in proposing microservices. Both examples are modified to generate more and less granular microservices. The coupling and cohesion values of the original examples are compared to those of the more and less granular microservices. We observe that domain-driven design has delivered a good end result for finding modular microservices.

INDEX TERMS Bounded context, cloud computing, domain-driven design, microservice architecture.

I. INTRODUCTION

Enterprise informatization has deepened over the years. As mentioned in [1], enterprise information systems were developed in ''chimney'' structures. These structures form a set of modules developed on top of the same framework. Due to the high coupling of the modules, these information systems have become large monoliths that are difficult to maintain and adapt to new cloud paradigms. In cloud information systems, service availability is critical. Service-level agreements require the cloud application to be resilient. A failure in one service should not affect the entire system. As a result, the modularity of the application components has become critical.

Microservice architecture (MSA) was proposed to achieve more modular and independently scalable components. The main idea behind microservices (MSs), based on the Unix philosophy, is that each service is responsible for performing one business task, that is, doing one thing and doing it well [2]. Microservices are supposed to be modular, loosely coupled, and highly cohesive structures. Each microservice is deployed in its own container and is run individually. There is no agreed-upon definition of a microservice; however, there are accepted characteristics, as mentioned in [3]:

- Smart endpoints rather than smart pipes: MSA prefers simple queue structures over an enterprise service bus (ESB).
- Componentization around services: A change in a component should not require a change in other services.
- Organization around business capabilities: To achieve easier enforcement of modularity, MSA prefers to design the application, and the organization, around business capabilities.
- Products not projects: The development team should also be responsible for running the service.
- Decentralized governance: Choreography is preferred over orchestration.
- Decentralized data management: Separate databases (DBs) are preferred for each microservice.
- Infrastructure automation: The continuous integration and continuous delivery pipelines should be automated.
- Design for failure: A failure in one service should not greatly affect any other services.
- Evolutionary design: The deployment of services should be easier and faster than that of a monolithic application.

The associate editor coordinating the review of this manuscript and approving it for publication was Christian Esposito.

In the cloud environment, even though resources are limited, users should not experience any limitations as long as they are willing to pay as required. Scalability is critical in achieving cost-effective cloud platforms. As a result, the scalability practice of monolithic applications is no longer desirable. Even service-oriented architecture (SOA) falls short of achieving the required cloud scalability due to the complexity of the ESB. Unlike SOA, communication between microservices is supposed to consist of ''dumb pipes'' (not complex, as in the ESB). Even though SOA and MSA look very similar, there are differences between them. As mentioned in [4], some people refer to microservices as SOAs done right. Given that microservices are a new architectural style, what are the guidelines for creating a microservice? As stated in [5], different techniques have been proposed for identifying the boundaries of a microservice. However, there are no specific agreed-upon guidelines.

Since the introduction of microservices, finding the optimal granularity of a microservice has been a challenge [3]. As mentioned in [6], different propositions have been made for the optimal granularity (size) of a microservice. Some of these definitions propose having no more than a certain number of lines of code, some suggest that the microservice should be small enough to rewrite in 6 weeks, and the Amazon definition suggests a maximum team size of two pizzas (two pizzas should be enough to feed the entire team). As mentioned in [7], it is easy to make a mistake when partitioning an application into microservices.

Some experts claim that in trying to adopt microservices, an antipattern called nanoservices has emerged [8]. There have also been studies on how to measure the size of a microservice [9], [10]. As mentioned in [11], the emphasis has not necessarily been on the size but rather on the business capabilities of the microservices.

Domain-driven design (DDD) is one of the proposed methods of extracting microservices from a domain, and according to [12], it evolved after a considerable number of discrepancies arose in the work of domain experts, analysts, designers, and developers. It became obvious that there was a need to construct a common language. DDD is best suited to complex systems with many different domain knowledge areas. It adds a cost for introducing well-defined domain boundaries. The different domain components are called bounded contexts. Each bounded context can be interpreted as a microservice.

DDD is a good starting point for identifying microservices; however, where the line for the bounded context should be drawn is open to interpretation [13]. It is well known that one of the greatest benefits of microservices is that they are loosely coupled [3]. Although there have been previous studies on using DDD to identify microservices, to the best of our knowledge, there have not been any studies examining whether DDD provides optimal granularity. Therefore, the challenge of where to draw the boundaries for the optimal bounded context remains [13].

The objective of this study is twofold: to investigate whether DDD leads to the optimal granularity of a microservice and to understand whether cohesion and coupling calculations can help identify the optimal bounded context. In this case study, we aim to obtain empirical answers to four research questions (RQ) to clarify the mentioned objectives:

- RQ1: If the size of a microservice is small, does this imply that it is loosely coupled?
- RQ2: If the size of a microservice increases, will the cohesion increase?
- RQ3: Do the minimum coupling value and the largest cohesion value define the optimal microservice granularity?
- RQ4: Can DDD lead to the optimal microservice granularity?

In a desirable modular microservice architecture, the coupling is expected to be low and the cohesion is expected to be high. The coupling measure focuses on evaluating the dependencies between modules and aims to reduce such dependencies, while the cohesion measure focuses on how related the functions in a module are. These two measures contradict each other; the former tries to combine the modules, while the latter tries to make them more granular [14]. Coupling and cohesion have been widely used to identify the optimal modularity (also called granularity in the microservices domain [15]) of software components [16]–[19]. To determine the optimal domain boundaries, a good starting point is to measure coupling and cohesion [20]. Drawing the boundary that maximizes cohesion and minimizes coupling could be the best way to define the service division. In this paper, the optimal microservice is defined as having a balance between the minimum coupling and the maximum cohesion values. Thus, RQ3 tries to get the optimum selection between coupling and cohesion measurements by applying the analytic hierarchy process (AHP). The goal of RQ1 and RQ2 are to determine whether coupling and cohesion changes smoothly based on the granularity (size). Is it possible to directly conclude that smaller modules always have less coupling? Since coupling and cohesion are conflicting measures, can we always expect a microservice with medium granularity to be the best case that provides the best balance between coupling and cohesion values? Or is there a sweet spot that is difficult to catch unless we measure carefully? RQ4 targets seeing if the selected DDD examples lead to low coupling and high cohesion.

The rest of this paper is organized as follows: Section 2 summarizes the relevant studies in the literature. Section 3 provides background about DDD, followed by the usage of coupling and cohesion measurements to find the optimal size of microservices. Section 4 explains the case studies carried out. The results are analyzed in Section 5, followed by a discussion in Section 6 and threats to validity in Section 7. Finally, concluding remarks can be found in Section 8.

II. LITERATURE REVIEW

This study aims to shed light on finding the optimal granularity of a microservice by measuring the static coupling and coherency values. Accordingly, two separate searches were performed on the Web of Science [21]: a microservice coupling search and a microservice coherency search. The former identifies research focusing on the coupling aspect of microservices, while the latter identifies research focusing on the coherency aspect of microservices.

The microservice coupling search was performed with the following keywords on August 3, 2019: ((microservice OR ''Micro service'') AND ((coupling OR coupled) OR couple)). The search resulted in 42 research papers. Of these, 30 papers focused on neither boundary identification (granularity) nor the effect of coupling on finding the optimal size. The remaining 12 papers are discussed below.

Several studies have applied model-driven design (MDD) to MSA. MSA MDD and SOA MDD are conceptually compared in [22]. Another study incorporated MDD into microservices and proposed using DDD to identify microservices [23]. Then, MDD was used to transform the domain models into more specific models that developers could better interpret during implementation. The proposed example in [23] had only two microservices, and there was no ambiguity concerning the bounded context. The researchers later incorporated aspect-oriented modeling technology [24] into their previous study [23]. This time, they used a more elaborate example. However, the way they determined their service boundaries was not covered in their study.

One real-life example related to generating microservices can be found in [25]. This study explains the best practices adopted for microservice usage in one of the largest e-commerce sites in Europe. To improve resiliency, they favor treating decoupling as a shared-nothing principle. They started with four loosely coupled applications and moved to 12 verticals. They defined a vertical as a part of the platform that is responsible for a single bounded context. According to their definitions, verticals are coarser grained than microservices. However, they did not clearly explain the differences between a microservice and a vertical or why they used verticals in addition to microservices.

Considering that microservices are a fairly new approach [26], [27], there have been multiple studies concerning dividing existing monolithic applications into microservices. One such study [28] suggested that existing services be decoupled by converting them into separately deployable components; the focus was not on revisiting the service boundaries. In [29], promise theory was used to form the microservices. The authors claimed that the formed microservices were loosely coupled, but no coupling measurement was included. Another study on moving from a monolith to microservices [30] mentioned that there are many factors relevant to dividing a monolithic application into microservices, such as functional dependency, functional popularity, and security. In their study, they divided the monolithic application so that each function was a different service. Their focus was on comparing the performance of the monolithic application to that of the microservices. They did not investigate other possible divisions, nor did they measure the coupling values of the resulting microservices.

Another study [7] focusing on the extraction of microservices from a monolithic application leveraged static and dynamic analyses. First, they generated a call graph using a static analysis. Next, by tracing data and function calls in real time, they generated the coupling degree of the functions and classes from the interaction frequency. Using the k-means clustering algorithm, they determined the partition (the granularity) of the microservices. In their study, they mentioned that the dynamic analysis method is limited to automated test coverage. In [2], the authors proposed an automated method for identifying possible microservices in a monolithic application. They measured the static coupling based on abstract syntax trees. Evolutionary coupling was measured by detecting changes between consecutive commits. Then, a graph clustering algorithm was used to identify the microservice candidates. The resulting candidates were examined by the developers to determine their cohesion values. However, no details were provided as to how the cohesion values were measured.

As mentioned above, there have been several detailed studies on the extraction of microservices from monolithic applications. Therefore, it is not surprising that there has also been a systematic mapping of the migration of legacy systems to microservices [31]. Studies have examined the challenges and quality attributes of microservices, as well as the motivation behind moving from a monolithic system to an MSA.

In addition to MDD, DDD has also been applied to MSAs. One example describes the architecture of a community healthcare system [32]. DDD was mentioned in the study; nonetheless, the main focus was not on how the division was established. One notable study concerning applying DDD to microservices is [33]. The authors tried three different methods to extract microservices from a monolithic application. The first was a logical coupling strategy. The coupling was measured based on the revision histories of the classes that changed together. The second extraction method was based on contributor coupling, which was calculated according to the classes that changed most due to the same contributors. The third method was semantic coupling, which was based on DDD. Of these three methods, DDD gave the best results.

Thus far, we have discussed the microservice coupling search results. Next, we discuss the microservice coherency search. This search was performed with the following keywords on August 07, 2019: ((Microservice OR ''micro service'') AND (coher OR coherency OR coherent OR cohesion OR cohesive)). The search resulted in 12 research papers, 4 of which were already evaluated as part of the previous microservice coupling search.

TABLE 1. Mapping the results of microservice coupling and cohesion searches.

Referenced papers	Implemented solution versus conceptual analysis	Migrating a legacy application	Creating/improving existing microservices
[23]	Conceptual	N/A	N/A
[29]	Conceptual	N/A	N/A
[24]	Implemented	Yes	No
[2]	Implemented	Yes	No
[7]	Implemented	Yes	No
[28]	Implemented	Yes	No
[31]	Implemented	Yes	No
[30]	Implemented	Yes	No
[33]	Implemented	Yes	No
[32]	Implemented	Yes	No
[35]	Implemented	Yes	No
[34]	Implemented	Yes	No
[37]	Implemented	Yes	No
[25]	Implemented	No	Yes
[22]	Implemented	No	Yes
[6]	Implemented	No	Yes
[36]	Implemented	No	Yes
Current Study	Implemented	No	Yes

Of the remaining eight research papers, three did not focus on boundary identification (granularity) or the effect of coherency on finding the optimal size. The remaining five papers are discussed below.

Similar to the studies mentioned as part of the microservice coupling search, the theme of converting legacy applications to microservices extends to the microservice coherency search results. One such study [34] used the software architecture finder (SArF) clustering algorithm to identify possible microservices from two different monolithic applications. The results for the first application were expected to be similar to the original microservices created for that application; however, the results did not match their expectations. In addition to clustering algorithms, promise theory has also been applied to microservice identification [35]; however, this study did not include information concerning the validation of their results via measurements of coherency.

Machine learning algorithms have also been applied to microservice identification. In [6], the genetic algorithm was applied to identify and improve microservices. This resulted in a 20% improvement; nevertheless, the evaluation criteria did not include coupling or cohesion. In [36], the authors proposed a method to identify microservices by evaluating the OpenAPI specification using DISCO, a database for identifying the semantic similarity of terms, and found that when the results were too coarse grained, they had to rerun the algorithm to improve the results.

In [37], the authors proposed a variability-based, pattern-driven architecture migration. Depending on the needs and goals of the company, certain patterns for migrating to the cloud are advised. This study primarily focused on the migration process rather than on how to divide a certain service into microservices.

In this study, we examined 12 coupling-related and 5 cohesion-related papers that primarily focused on the migration of existing legacy code into MSA. Only four papers were related to writing a microservice from scratch or improving an existing microservice. Of these, two papers discussed concepts but did not operate on real-life examples with empirical results.

As shown in Table 1, hands-on research concerning starting with microservices and creating a new application has not yet been conducted in detail. A total of 73% of hands-on studies (4 out of 15 implementation papers), excluding the current study, focused on the migration of legacy applications. Accordingly, the current study aims to generate relevant answers at design time. As shown in [33], DDD is a good means of identifying microservices at design time. However, the problem of defining the bounded context is not clear and is open to study. Accordingly, this study proposes measuring the coupling and coherency values to identify the best possible bounded context. The last row in Table 1 indicates that the current study is one of the few examples of an empirical study focusing on creating microservices from scratch or improving existing microservices.

III. BACKGROUND

A. DOMAIN-DRIVEN DESIGN

DDD is an approach to software development for complex needs that connects implementation to an evolving model [38], and according to [12], it evolved after considerable discrepancies arose in the work of domain experts, analysts, designers, and developers. It became obvious that there was a need to construct a common language, called the ''ubiquitous language'' in DDD.

The main feature of DDD is that it puts the domain knowledge at the center of the design. There are four layers in DDD:

- User interface: This is the presentation layer of the design.
- Application layer: The main goal of the application layer is to provide a means of communication between the user layer and the domain layer. There is no logic at this level.
- Domain layer: This is the heart of the design. All the domain-related functionality resides in this layer.
- Infrastructure layer: This layer serves all the infrastructure needs of the other layers, such as common libraries that are not part of the domain knowledge.

The building blocks of DDD are as follows [39]:

- Entity: Objects that have an identity are called entities.
- Value object: When an object does not have an identity and is useful only for having certain attributes or performing a calculation, then it is called a value object. Value objects are immutable.
- Domain event: Unlike the other building blocks, domain events were introduced after the original DDD book [12] was published. A domain event represents an event that the domain expert (the end user) cares about.
- Aggregate: When certain objects are related to each other in the sense that one of them cannot exist without the other (the root), then these objects are combined into a single identity called an aggregate. The main purpose of aggregates is to consolidate the domain knowledge into a single object so that no external methods can change it incorrectly.
- Service: When an operation does not belong to only one entity or aggregate, then it is best to implement that functionality as a service.
- Repository: The infrastructure is responsible for carrying out persistence operations (e.g., reading from a database). The repository pattern shields the details of the persistence in such a way that the aggregates can be properly encapsulated.
- Factory: There are cases in which the construction of an object is so complicated that the caller might become entangled in strongly coupled code. To avoid such cases, factories should be leveraged. Using the factory pattern frees the caller from dealing with the details of the object to be used.

DDD is best suited for complex systems with many different domain knowledge areas. It adds a cost for introducing well-defined domain boundaries. The different domain components are called the bounded context. The main advantages of DDD are maintainability, the autonomy of the team, the ease of continuous integration, and the use of different persistence tools. However, DDD comes with the price of complex design and a costlier initial development. In a cloud business, where changes occur on a continuous basis, the initial development time is less of a concern than maintainability. As a result, some experts advise the adoption of DDD along with microservices in cloud systems [40].

B. OPTIMAL GRANULARITY OF A MICROSERVICE

Microservices are becoming the new architectural style for cloud applications due to their advantages, such as being easier to maintain, being more adaptable, and complying with the needs of continuous integration, as the system downtime is almost zero [41]. On the other hand, microservices also come with extra costs, such as distributed communication. Microservices are self-contained processes that talk over the network. Distributed communication introduces extra complexity and possible performance issues.

When the size of the software grows, interdependency and coupling generally increase [42]. A big monolith application is expected to have higher coupling and low cohesion values. On the contrary, when a particular service is divided into multiple microservices, serialization and deserialization costs, remote network calls, and possible failures increase. Some aspects are better when the services are less granular. On the other hand, some aspects would benefit from more granular services. Finding the optimum size for a service would mean finding the sweet spot that balances all different aspects to consider.

There has been research around extracting microservices from monolith legacy applications [2], [7], [30]. A considerable amount of such research focuses on measuring static code analysis metrics such as coupling and cohesion [2], [7], [16], [17], [19]. The optimum result was targeted through low coupling and high cohesion values. In order to measure the coupling values, afferent and efferent coupling can be used. Cohesion can be measured through relational cohesion. Some of these research studies had promising results when DDD was applied to extract the microservices [33].

DDD suggests that the software should be designed around domain knowledge. Each bounded context represents a microservice. However, the bounded context is a logical term defined as 'part of the software where particular terms, definitions, and rules apply consistently.' Such a logical division entails ambiguity in deciding on the boundaries of the bounded context [12], [13].

The current study focuses on determining the bounded context using coupling and cohesion measurements, given that there is ambiguity around finding the right bounded context. To avoid any possible bias, we opted to select cases already modularized based on DDD, which we call the original examples. Each example is restructured to generate fewer microservices (less granular microservices (LGMs)) and more microservices (more granular microservices (MGMs)).

Next, coupling and cohesion measurements are taken for the newly generated examples and for the original. Based on the coupling and cohesion measurements, the best-performing microservice construction is identified as the solution with the optimal granularity. The analytic hierarchy process (AHP) is applied to select the best microservice.

IV. CASE STUDIES

To answer the abovementioned research questions, we carried out a detailed literature review to find suitable original examples to start with, instead of developing our own examples. Because we aimed at selecting examples that are already designed by applying DDD and coded. We believe that such open examples are more appropriate for evaluation; thus, they will be accessible for any reader. We came across multiple examples during this research; some of them did not share the code or the design diagram [13], [43]–[45]. Several DDD examples shared either implementation or UML diagrams. At our initial attempt, we chose [46] and created less granular microservices for it. However, creating a more granular microservice was not possible with that example, given that the original services were so simple to start with. Some of the other possible examples only had three to four simple microservices in them [47], or the UML class diagram did not include all the bounded-contexts in the system [24]. In the end, we selected [48], [49] as original examples because they both had complex enough microservices that could be divided into more granular ones.

The first is based on unified modeling language (UML) class diagrams that were generated in a previous study [48]. The second is based on an existing microservice implementation: eShopOnContainers [49]. In both examples, the original proposed solutions were generated by applying DDD concepts.

The coupling is measured via afferent coupling and efferent coupling, as covered in [50]. Afferent coupling is measured at the assembly level (the bounded context level for the UML class diagrams). This coupling calculates the number of types outside the assembly that depend on the types within the current assembly. High afferent coupling indicates that there are many other assemblies relying on the current assemblies. It is desirable to have low afferent coupling. The average afferent coupling values are calculated as shown in Equation (1).

Let $(C_a)_i$ denote the afferent coupling of an assembly i, where n is the number of assemblies.

$$Avg(C_a) = \frac{\sum_{i=1}^{n}(C_a)_i}{n} \tag{1}$$

Efferent coupling is measured by calculating the number of types outside the current assembly that the types within the current assembly use. If the efferent coupling of an assembly is high, this indicates that the assembly relies on many other assemblies. In an ideal world, efferent coupling is expected to have a low value. The average efferent coupling values are calculated as shown in Equation (2).

Let $(C_e)_i$ denote the efferent coupling of an assembly i, where n is the number of assemblies.

$$Avg(C_e) = \frac{\sum_{i=1}^{n}(C_e)_i}{n} \tag{2}$$

In this case study, cohesion is measured as relational cohesion within assemblies, as mentioned in [50]. It is desirable to have high relational cohesion. The relational cohesion (H) of an assembly is calculated as shown in (3).

Let R be the number of relationships among the types within the assembly, and let N be the number of types within the assembly.

$$H = \frac{R+1}{N} \tag{3}$$

The average relational cohesion values are calculated as shown in (4).

Let H_i denote the relational cohesion of an assembly i, where n is the number of assemblies.

$$Avg(H) = \frac{\sum_{i=1}^{n} H_i}{n} \tag{4}$$

A. CASE STUDY 1

To illustrate the challenges of DDD-based microservice design, a cargo-related domain model is used as an example in [48]. In the study, the authors included a UML class diagram of a cargo domain model with DDD-specific stereotypes. In the current case study, their UML class diagram is taken as the original DDD-based microservice example, as shown in Fig. 1. The UML diagram has three different bounded contexts ('Cargo', 'Customer', and 'Location'), which can be interpreted as microservices [48]. 'Cargo Bounded-Context' focuses on the cargo itself, the delivery specification, and the handling events that go along with the cargo. Cargo has a goal (a Delivery Specification with an arrival time). Delivery History holds Handling Events, some of which involve movement from one location to another. Cargo may have multiple Customers. The Customer Bounded Context focuses on the Customer and the Repository. The Customer can either be a sender or a receiver based on its role. The Location Bounded Context relies on the location service for supplying the destination location.

The original example is divided into smaller constructs, as shown in Fig. 2. The resulting example is called the MGM example. Previous studies suggest that the main difference between SOAs and microservices is the size and coupling [51]. We aim to analyze the effect of this design modification on the coupling and cohesion values.

The original example is combined into larger constructs to obtain another design alternative, as shown in Fig. 3. The resulting example is called the less granular microservice (LGM) example. If there is a relationship between the granularity and the coupling, there can be a relationship between the granularity and the cohesion. The aim of this design is to observe the cohesion and coupling values when the granularity of the services is reduced.

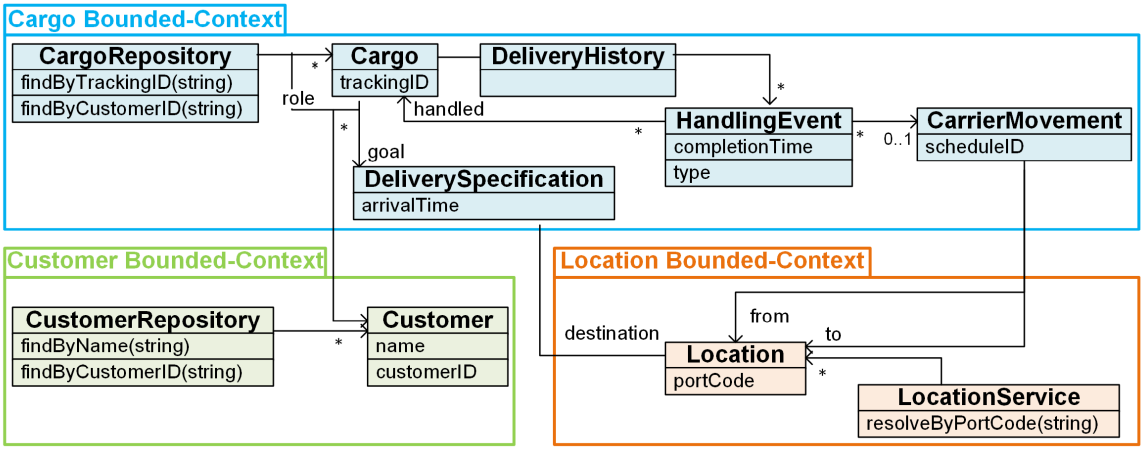

FIGURE 1. The UML class diagram for the original DDD-based cargo example.

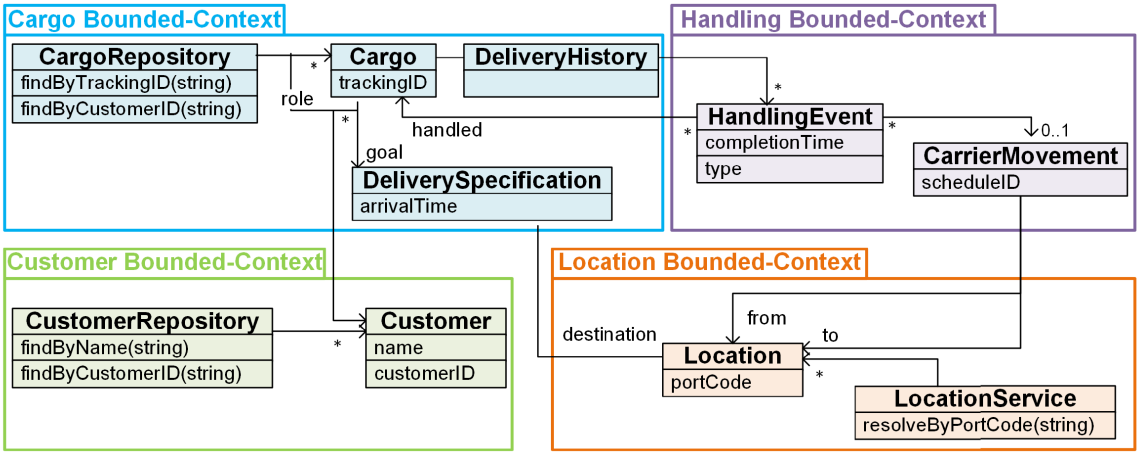

FIGURE 2. The UML class diagram for the MGMs of the cargo example.

To measure the afferent coupling, the incoming associations to the related bounded context are measured. A nondirectional association is considered to be a two-way association. The efferent coupling is measured as the number of outgoing associations for the related bounded context. To measure the relational cohesion, we use (3). The average of the resulting coupling values is shown in Fig. 4.

Based on the results, both the afferent and efferent coupling values are smallest for the original sample. The coupling value is the worst (highest coupling) for the LGM.

As shown in Fig. 5, the cohesion values increase as the size of the bounded context increases. The LGM has the best relational cohesion, followed closely by the original example.

B. CASE STUDY 2

In this case study, a well-known microservice example is selected [49] (i.e., eShopOnContainers, an e-commerce web application for online shopping).

The selected example is a microservice application running on Docker that was constructed by applying DDD concepts [52]. The eShopOnContainers example has a total of seven microservices. Of the seven services, four are of greatest interest in the current study and are shown in Fig. 6.

In this case study, the original example is modified to obtain two new examples, LGM and MGM examples (as described in Table 2). The code for the newly generated applications has been uploaded to GitHub for public use in future studies with the MIT license. Ultimately, the coupling and cohesion values of the three examples are measured to evaluate the success of each example. To create a more granular example, the original eShopOnContainers application was modified by dividing the order microservice into two microservices: the order and the buyer. The code for the MGM application is available on GitHub [53]. This division adds one more microservice to the original example, bringing the total number of microservices in the MGM example to eight. The less granular example combines the order and payment microservices. The code for the LGM application is also available on GitHub [54]. This modification results in one fewer microservice compared to the original example, reducing the total number of microservices in the LGM example to 6. The NDepend tool [55], which is one of the

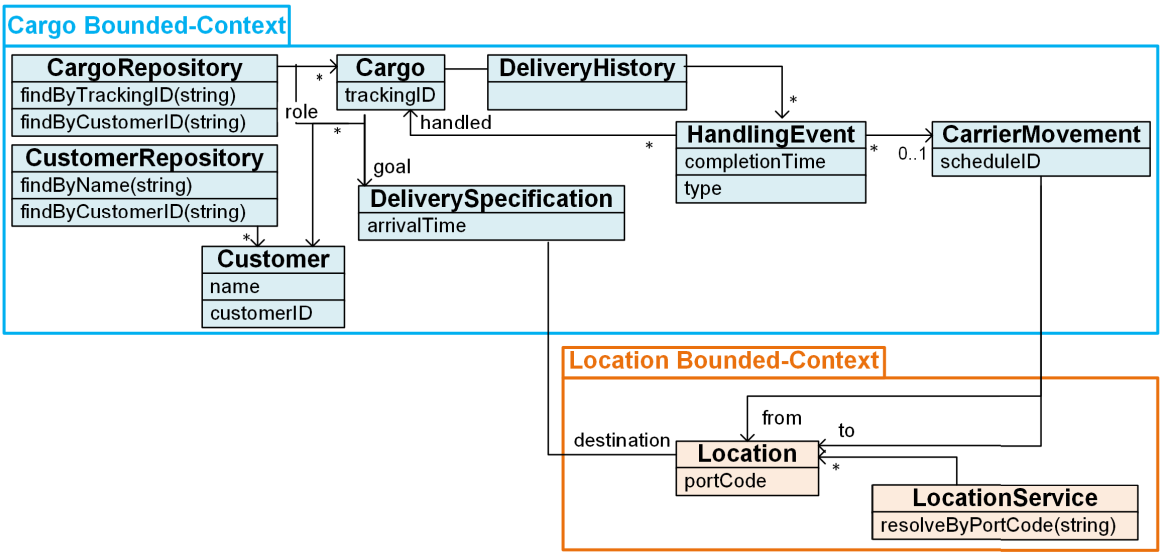

FIGURE 3. The UML class diagram for the LGMs of the cargo example.

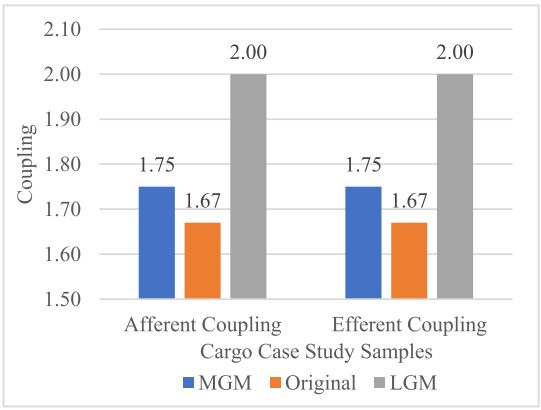

FIGURE 4. The afferent coupling and efferent coupling for each sample in the cargo case study.

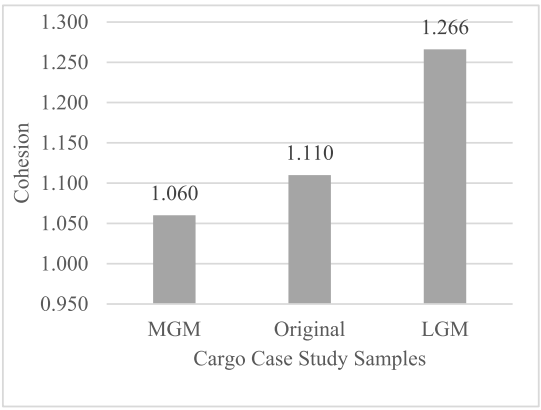

FIGURE 5. The relational cohesion values for each sample in the cargo case study.

TABLE 2. Services for the MGM, Original, and LGM Applications.

Application name	Names of the services	Total # of services
MGM	Basket, Catalog, Identity, Location, Marketing, Order, Buyer, Payment	8
eShopOriginal	Basket, Catalog, Identity, Location, Marketing, Order, Payment	7
LGM	Basket, Catalog, Identity, Location, Marketing, Order	6

most widely used tools for static code analysis, is used to calculate the static code analysis values.. Given that approximately 450 types exist in the example applications, using an automated static code analysis tool helped to minimize possible human error.

We measured the three metrics for all three example microservice applications: the MGM, the original eShopOnContainers application, which was constructed following the DDD concepts, and the LGM. The results are shown in Figs. 7 and 8 in terms of the average afferent coupling, average efferent coupling, and average relational cohesion.

The average afferent coupling values are 7.84 for MGMs, 7.45 for the original application, and 7.50 for LGMs. The average efferent coupling is 110.8 for MGMs, 107.5 for the original application, and 114.1 for LGMs. As shown in Fig. 7, the coupling values are the smallest for the original microservice application. The afferent coupling value is the largest for MGMs, whereas the efferent coupling value is the largest for LGMs. As shown in Fig. 8, the average relational cohesion value is 1.234 for MGMs, 1.265 for the original application, and 1.266 for LGMs. The relational cohesion values increase as the application becomes less granular. The difference between the original example and MGMs is more

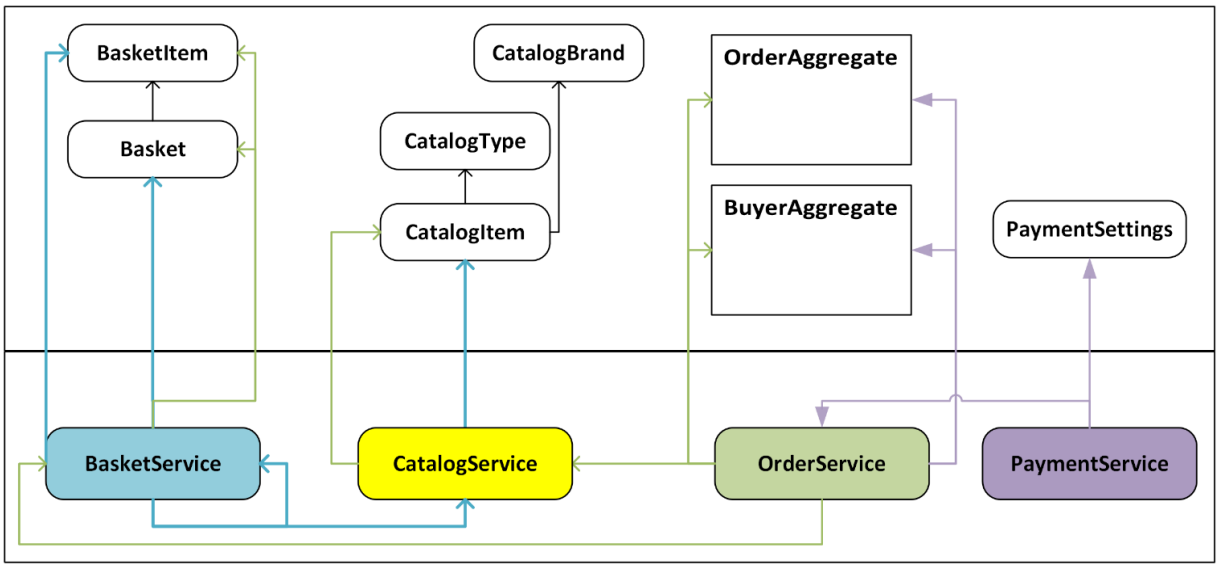

FIGURE 6. The UML class diagram for the MGMs of the eShopOnContainers example.

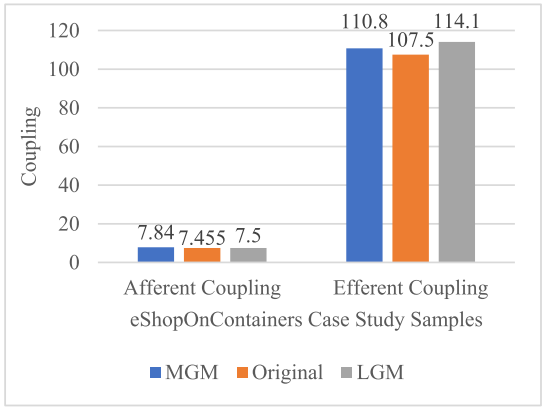

FIGURE 7. The average afferent and efferent coupling values for each sample in the eShopOnContainers case study.

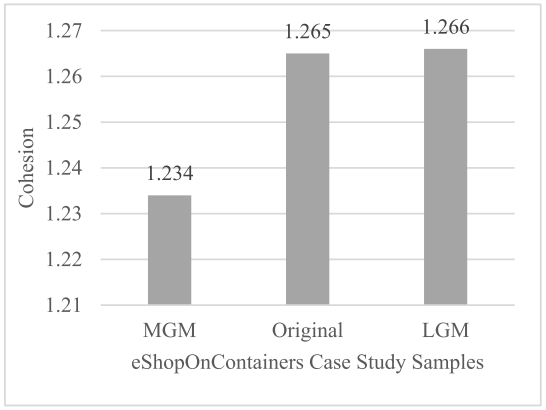

FIGURE 8. The relational cohesion values for each sample in the eShopOnContainers case study.

noticeable than the difference between the original example and LGMs.

V. ANALYSIS OF MEASUREMENTS

The two original case studies (cargo & eShopOnContainers) are updated to generate more granular and less granular microservices, resulting in the original, MGM and LGM applications. The afferent coupling, efferent coupling and relational cohesion values are measured for all the resulting microservice applications (3 applications for each case study). We first examine the cargo case study and then the eShopOnContainers case study. The main goal is to identify the optimal result based on the coupling and cohesion measurements. The final problem is a multiobjective decision-making (MCDM) problem. To perform MCDM, the analytic hierarchy process (AHP) [56] is carried out. Before carrying out the AHP, the values are normalized by applying min-max scalar normalization [57], [58].

Given a set of matching scores $\{s_k\}$, k = 1, 2, …, n, the normalized scores are given by (5), where *min* denotes the minimum value in the dataset and *max* denotes the maximum value in the dataset.

$$s'_k = \frac{s_k - min}{max - min} \tag{5}$$

The most desirable value in a column is assigned 1, and the least desirable value is assigned 0. For example, the relational cohesion is expected to be high. As a result, the LGM cargo application with the largest relational cohesion value is assigned 1. The MGM cargo application has the least relational cohesion, so it is assigned 0. The original cargo application has a value in between, closer to the desirable side. After applying the normalization formula, based on direct proportionality, it obtains a value of 0.71. The normalized numbers for each measurement are shown in Table 3.

TABLE 3. Normalized measurements for the cargo case study.

	Afferent Coupling	Efferent Coupling	Relational Cohesion
MGM Cargo	0.76	0.76	0
Original Cargo	1	1	0.71
LGM Cargo	0	0	1

To find the optimal result, we apply the AHP after normalization. As stated in [59], unless the importance of each measured aspect can be identified, the aspects are treated equally. This principle is called the principle of equal importance of features. As a result, all the weights are assigned equally in applying the AHP. To calculate the AHP values, an online calculator is utilized [60].

Table 4 shows that the most desirable result is achieved by the original cargo case study (having rank 1). The next most desirable solution is LGM cargo. The least optimal one is MGM cargo.

The measurements for eShopOnContainers applications are also normalized by applying min-max scalar normalization. The normalized numbers for eShopSmall, eShopOriginal, and eShopBig are shown in Table 5.

As shown in Table 6, the optimal result is achieved by the original eShop case. It is closely followed by the LGM eShop.

VI. DISCUSSION

Due to the strong advantages of MSA, most enterprises tend to prefer microservices [31]. Given that MSA is still new compared to SOA, there is no industry standard addressing how to construct microservices. However, DDD is seen as one of the most common methods of construction [61]. The challenge in using DDD is to identify the bounded context. Calculating the coupling and coherency values can provide a good source of guidance to overcome this challenge.

As part of the current study, three measurements are collected. With those measurements, the expected behavior of a definition of good design is explained as follows:

- Afferent Coupling: The afferent coupling values are expected to be low.
- Efferent Coupling: The efferent coupling values are expected to be low.
- Relational Cohesion: Cohesion is expected to be high.

As previously mentioned, the current study aims to answer four research questions. In light of the findings in the previous section, the answers to those research questions are given below.

– RQ1: If the size of a microservice is small, does this imply that it is a loosely coupled microservice?

Both MGM and LGM had larger coupling values compared to the original case. One of the greatest benefits of MSA is reducing coupling. Moving from a monolith to smaller services is expected to result in less coupling [30]. Some people refer to microservices as SOA done right [61]. However, as shown in Figs. 4 and 7, smaller microservices do not always result in less coupled code. According to our results, there is a sweet spot in finding the optimal coupling value for a service.

– RQ2: If the size of a microservice increases, will the cohesion increase?

In these particular examples, it appears that the cohesion increased due to the increase in the microservice size. However, two examples are not sufficient to justify a general rule. If all the services are combined into one monolithic service, this could affect the cohesion in a negative way. More examples need to be examined before a definite answer is reached.

– RQ3: Do the minimal coupling value and the maximal cohesion value define the optimal microservice granularity?

As shown in Figs. 4 and 7, the coupling values are best in the original example. According to Fig. 8, the cohesion value is best for the LGMs. However, the increase in cohesion is not very noticeable compared to the small decrease in coupling shown in Fig. 7. In both the Cargo and the eShopOnContainers examples, the original applications had the best coupling. However, the cohesion values were the highest in the LGM applications. The choice is thus not as easy as ''just go with the middle result''. To solve this multiobjective decision-making problem, we decided to use AHP. After carrying out the AHP in both case studies, the original application emerged as the first chosen option for the optimal result. Therefore, the optimal point for granularity is the original example, which has the desired loose coupling and acceptable cohesion values. The optimal granularity point can be seen as the best place to identify the bounded context. Identifying the optimal bounded context is a challenge when using DDD [13]. Therefore, the coupling and cohesion values can be used to identify the optimal bounded context in DDD.

– RQ4: Can DDD lead to the optimal microservice granularity?

In the given case studies, the optimal granularity of the microservices is achieved in the original cases. In both studies, the original examples were created by following DDD concepts. Therefore, for these particular examples, it is safe to say that DDD leads to the optimal level of granularity.

As previously mentioned, there are two objectives of this study. We found the following:

– The problem of identifying the optimal bounded context can be solved by measuring the coupling and cohesion values and choosing the least coupled and most cohesive division point as the bounded context. To calculate the optimal point, which is the least coupled and most cohesive point, the AHP can be used.
– Using DDD leads to the optimal granularity; therefore, the ongoing debate surrounding the optimal size of a microservice can be addressed by applying DDD.

TABLE 4. AHP results for the cargo case study.

	Afferent Coupling	Efferent Coupling	Relational Cohesion	Weighted Sum	Rank
Criteria Weight	0.333	0.333	0.333	-	-
MGM Cargo	0.76	0.76	0	0.507	2
Original Cargo	1	1	0.71	0.903	1
LGM Cargo	0	0	1	0.333	3

TABLE 5. Normalized measurements for the eShopOnContainers case study.

	Afferent Coupling	Efferent Coupling	Relational Cohesion
MGM eShop (eShopSmall)	0	0.51	0
Original eShop (eShopOriginal)	1	1	0.97
LGM eShop (eShopBig)	0.88	0	1

TABLE 6. AHP results for the eShop case study.

	Afferent Coupling	Efferent Coupling	Relational Cohesion	Weighted Sum	Rank
Criteria Weight	0.333	0.333	0.333	-	-
MGM eShop	0	0.51	0	0.17	3
Original eShop	1	1	0.97	0.99	1
LGM eShop	0.88	0	1	0.627	2

VII. THREAT TO VALIDITY

MSA primarily uses events and queue mechanisms [49]. Communication among services is carried out via asynchronous events sent through a queue mechanism [62]. An event is a value added to the queue by a certain service that will be consumed by a different service. Static code analysis can identify the call to the queue and the invocation from the queue; however, it misses the value of the event. As a result, static code analysis can identify the coupling and cohesion values between a service and a queue but will miss the hidden dependencies among certain services. Given that some MSAs use asynchronous event mechanisms, static analysis alone might not be sufficient. Adding measurements based on usage patterns and data flow could provide more realistic coupling and cohesion values.

Currently, most applications use object relational mapping (ORM) for DB access. Using ORM means having items in the DB represented as classes in the code. Given that static code analysis tools inspect the classes and calls in a code, when ORM is used, a static code analysis will obtain more concise numbers for coupling and cohesion. However, when an application carries out direct DB calls rather than using ORM, a static code analysis will miss the coupling information at the DB level. In our case study, the code implementation used an Entity Framework Core as the ORM. As a result, this was not an issue. However, in future studies that do not use ORM, coupling and cohesion measurements at the code level will not be sufficient.

VIII. CONCLUSION

In this study, our goal was to investigate an appropriate means of identifying the optimal bounded context and the optimal granularity of a microservice. To achieve this, two sample DDD-driven applications were selected as the original cases. For each example, the original application was divided into MGMs and combined into LGMs to construct additional applications. Then, the average afferent coupling, efferent coupling, and relational cohesion values at the class level were measured. The results were compared by applying the AHP to answer four research questions. The following conclusions emerged based on the empirical results:

- The use of DDD proved to be a good means of identifying the optimal microservice granularity.
- Calculating the coupling and cohesion values helped to identify the optimal bounded context.

Currently, hands-on research on microservice granularity and division primarily focuses on dividing legacy microservices (73%). We expect that the number of studies focusing on writing microservices from scratch will increase over time. Therefore, measurements at design time will become increasingly useful in the future.

Even though the literature mentions that the microservices have a granularity problem (finding the right size) [15], we believe that it is actually a modularity problem. There is no easy answer for finding the right size of a microservice where there are so many different factors, such as complexity of the domain, the measurement method, the language being used etc. In our small number of examples, we observed that domain-driven design has delivered a good end result for finding modular microservices. More research should be carried out as more examples become available to support a concluding remark.

Although two case studies are not sufficient for making a sweeping generalization, they do provide a good starting point for future elaboration. Given that the microservices architecture is still a fairly new concept, it is difficult to find case studies that construct the microservices based on DDD. As more examples become available, this study can be repeated with a classification of multiple case studies based on a specific metric. More studies and results are needed to draw a definite conclusion as to whether ''DDD gives the optimum modularity in all cases''. Moreover, improvements can be made to the proposed method by investigating whether measurements concerning asynchronous eventing behavior will be useful. For example, the eShopOnContainers example includes integration events (across microservices) and domain events (within a microservice). These events may be relevant to measuring the coupling and cohesion values from an asynchronous microservice perspective. In future studies, the proposed method will be expanded by adding measurements for asynchronous eventing behavior.

In future studies, we plan to explore different approaches focusing on identifying the optimal bounded context at design time. To keep up with the ever-changing requirements of customers and the environment, code in the cloud needs to change at a fast pace. Having the right design from the beginning is no longer possible. The granularity and efficiency of microservices need to change over time. A monitoring tool for measuring microservice granularity metrics over time and giving suggestions for code refactoring would be beneficial to everyone who adopts MSA.

REFERENCES

[1] Y. Xie, X. Zhou, H. Xie, G. Li, and Y. Tao, ''Research on the architecture and key technologies of integrated platform based on micro service,'' in *Proc. IEEE 3rd Adv. Inf. Technol., Electron. Autom. Control Conf. (IAEAC)*, Chongqing, China, Oct. 2018, pp. 887–893.

[2] S. Eski and F. Buzluca, ''An automatic extraction approach: Transition to microservices architecture from monolithic application,'' in *Proc. 19th Int. Conf. Agile Softw. Develop., Companion*, Porto, Portugal, May 2018, pp. 1–6.

[3] J. Lewis and M. Fowler. *Microservices*. Accessed: Oct. 10, 2019. [Online]. Available: https://martinfowler.com/articles/microservices.html

[4] M. Castro. *SOA Done Right: Microservices and Microservice Architecture*. CODE Magazine. Accessed: Oct. 30, 2020. [Online]. Available: https://www.codemag.com/Article/1801081/Understanding-Microservices-and-Microservice-Architecture

[5] Z. Li, D. Seco, and A. Sánchez Rodríguez, ''Microservice-oriented platform for Internet of big data analytics: A proof of concept,'' *Sensors*, vol. 19, no. 5, p. 1134, Mar. 2019, doi: 10.3390/s19051134.

[6] S. Klock, J. M. E. M. van der Werf, J. P. Guelen, and S. Jansen, ''Workload-based clustering of coherent feature sets in microservice architectures,'' in *Proc. IEEE Int. Conf. Softw. Archit. (ICSA)*, Gothenburg, Sweden, Apr. 2017, pp. 11–20.

[7] Z. Ren, W. Wang, G. Wu, C. Gao, W. Chen, J. Wei, and T. Huang, ''Migrating Web applications from monolithic structure to microservices architecture,'' in *Proc. 10th Asia–Pacific Symp. Internetware*, Beijing, China, Sep. 2018, pp. 1–10.

[8] A. Rotem-Gal-Oz. *Services, Microservices, Nanoservices—Oh My!* Cirrus Minor. Accessed: Oct. 14, 2019. [Online]. Available: https://arnon.me/2014/03/services-microservices-nanoservices/

[9] G. Schermann, J. Cito, and P. Leitner, ''All the services large and micro: Revisiting industrial practice in services computing,'' in *Service-Oriented Computing—ICSOC 2015* (Lecture Notes in Computer Science), A. Norta, W. Gaaloul, G. Gangadharan, and H. Dam, Eds. Berlin, Germany: Springer, 2016, pp. 36–47.

[10] H. Vural, M. Koyuncu, and S. Misra, ''A case study on measuring the size of microservices,'' in *Computational Science and Its Applications—ICCSA 2018* (Lecture Notes in Computer Science), B. Murgante, S. Misra, E. Stankova, C. M. Torre, O. Gervasi, A. M. A. Rocha, D. Taniar, B. O. Apduhan, E. Tarantino, and Y. Ryu, Eds. Cham, Switzerland: Springer, 2018, pp. 454–463.

[11] C. de la Torre, B. Wagner, and M. Rousos. *Identifying Domain-Model Boundaries for Each Microservice*. Microsoft Docs. Accessed: Oct. 14, 2019. [Online]. Available: https://docs.microsoft.com/en-us/dotnet/architecture/microservices/architect-microservice-container-applications/identify-microservice-domain-model-boundaries

[12] E. Evans, *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Boston, MA, USA: Addison-Wesley, 2004.

[13] M. I. Josélyne, D. Tuheirwe-Mukasa, B. Kanagwa, and J. Balikuddembe, ''Partitioning microservices: A domain engineering approach,'' in *Proc. Int. Conf. Softw. Eng. Afr. (SEiA)*, Gothenburg, Sweden, May/Jun. 2018, pp. 43–49.

[14] I. Candela, G. Bavota, B. Russo, and R. Oliveto, ''Using cohesion and coupling for software remodularization: Is it enough?'' *ACM Trans. Softw. Eng. Methodol.*, vol. 25, no. 3, pp. 1–28, Aug. 2016, doi: 10.1145/2928268.

[15] S. Hassan, R. Bahsoon, and R. Kazman, ''Microservice transition and its granularity problem: A systematic mapping study,'' *Softw., Pract. Exper.*, vol. 50, no. 9, pp. 1651–1681, Sep. 2020, doi: 10.1002/spe.2869.

[16] S. Mancoridis, B. S. Mitchell, C. Rorres, Y. Chen, and E. R. Gansner, ''Using automatic clustering to produce high-level system organizations of source code,'' in *Proc. 6th Int. Workshop Program Comprehension*, Ischia, Italy, Jun. 1998, pp. 45–52.

[17] M. Paixao, M. Harman, Y. Zhang, and Y. Yu, ''An empirical study of cohesion and coupling: Balancing optimization and disruption,'' *IEEE Trans. Evol. Comput.*, vol. 22, no. 3, pp. 394–414, Jun. 2018, doi: 10.1109/TEVC.2017.2691281.

[18] P. C. Jha, V. Bali, S. Narula, and M. Kalra, ''Optimal component selection based on cohesion & coupling for component based software system under build-or-buy scheme,'' *J. Comput. Sci.*, vol. 5, no. 2, pp. 233–242, Mar. 2014, doi: 10.1016/j.jocs.2013.07.003.

[19] X. Zhang, A. Luo, Y. Mao, M. Lin, Y. Kou, and J. Liu, ''A semi-automatic optimization design method for SvcV-5 in DoDAF 2.0 based on service identification,'' *IEEE Access*, vol. 8, pp. 33442–33460, Jan. 2020, doi: 10.1109/ACCESS.2020.2970446.

[20] K. Praditwong, M. Harman, and X. Yao, ''Software module clustering as a multi-objective search problem,'' *IEEE Trans. Softw. Eng.*, vol. 37, no. 2, pp. 264–282, Mar. 2011, doi: 10.1109/TSE.2010.26.

[21] Clarivate Analytics. *About us*. Accessed: Oct. 10, 2019. [Online]. Available: https://apps.webofknowledge.com

[22] F. Rademacher, S. Sachweh, and A. Zundorf, ''Differences between model-driven development of service-oriented and microservice architecture,'' in *Proc. IEEE Int. Conf. Softw. Archit. Workshops (ICSAW)*, Gothenburg, Sweden, Apr. 2017, pp. 38–45.

[23] F. Rademacher, J. Sorgalla, P. N. Wizenty, S. Sachweh, and A. Zündorf, ''Microservice architecture and model-driven development: Yet singles, soon married (?),'' in *Proc. 19th Int. Conf. Agile Softw. Develop., Companion*, Porto, Portugal, May 2018, pp. 1–5.

[24] F. Rademacher, S. Sachweh, and A. Zundorf, ''Aspect-oriented modeling of technology heterogeneity in microservice architecture,'' in *Proc. IEEE Int. Conf. Softw. Archit. (ICSA)*, Hamburg, Germany, Mar. 2019, pp. 21–30.

[25] W. Hasselbring and G. Steinacker, "Microservice architectures for scalability, agility and reliability in E-commerce," in *Proc. IEEE Int. Conf. Softw. Archit. Workshops (ICSAW)*, Gothenburg, Sweden, Apr. 2017, pp. 243–246.

[26] A. Carrasco, B. V. Bladel, and S. Demeyer, "Migrating towards microservices: Migration and architecture smells," in *Proc. 2nd Int. Workshop Refactoring*, Montpellier, France, Sep. 2018, pp. 1–6.

[27] H. Vural, M. Koyuncu, and S. Guney, "A systematic literature review on microservices," in *Computational Science and Its Applications—ICCSA 2017*, O. Gervasi, B. Murgante, S. Misra, G. Borruso, C. M. Torre, A. M. A. C. Rocha, D. Taniar, B. O. Apduhan, E. Stankova, and A. Cuzzocrea, Eds. Cham, Switzerland: Springer, 2017, pp. 203–217.

[28] S. Sarkar, G. Vashi, and P. P. Abdulla, "Towards transforming an industrial automation system from monolithic to microservices," in *Proc. IEEE 23rd Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, Turin, Italy, Sep. 2018, pp. 1256–1259.

[29] A. Tarmizi and M. Shanudin, "A method for analyzing and designing microservice holistically," *Int. J. Adv. Comput. Sci. Appl.*, vol. 8, no. 12, pp. 281–287, 2017, doi: 10.14569/ijacsa.2017.081236.

[30] V. Singh and S. K. Peddoju, "Container-based microservice architecture for cloud applications," in *Proc. Int. Conf. Comput., Commun. Autom. (ICCCA)*, Greater Noida, India, May 2017, pp. 847–852.

[31] S. Ahmed, A. Razzaq, S. Ullah, and S. Ahmed, "Matrix clustering based migration of system application to microservices architecture," *Int. J. Adv. Comput. Sci. Appl.*, vol. 9, no. 1, pp. 284–296, 2018, doi: 10.14569/ijacsa.2018.090139.

[32] R. Hill, D. Shadija, and M. Rezai, "Enabling community health care with microservices," in *Proc. IEEE Int. Symp. Parallel Distrib. Process. with Appl. IEEE Int. Conf. Ubiquitous Comput. Commun. (ISPA/IUCC)*, Guangzhou, China, Dec. 2017, pp. 1444–1450.

[33] G. Mazlami, J. Cito, and P. Leitner, "Extraction of microservices from monolithic software architectures," in *Proc. IEEE Int. Conf. Web Services (ICWS)*, Honolulu, HI, USA, Jun. 2017, pp. 524–531.

[34] M. Kamimura, K. Yano, T. Hatano, and A. Matsuo, "Extracting candidates of microservices from monolithic application code," in *Proc. 25th Asia–Pacific Softw. Eng. Conf. (APSEC)*, Nara, Japan, Dec. 2018, pp. 571–580.

[35] A. T. A. Ghani and M. Shanudin, "Method for designing scalable microservice-based application systematically: A case study," *Int. J. Adv. Comput. Sci. Appl.*, vol. 9, no. 8, pp. 125–135, 2018, doi: 10.14569/IJACSA.2018.090817.

[36] L. Baresi, M. Garriga, and A. D. Renzis, "Microservices identification through interface analysis," in *Service-Oriented and Cloud Computing*, F. D. Paoli, S. Schulte, and E. B. Johnsen, Eds. Cham, Switzerland: Springer, 2017, pp. 19–33.

[37] P. Jamshidi, C. Pahl, and N. C. Mendonça, "Pattern-based multi-cloud architecture migration," *Softw., Pract. Exper.*, vol. 47, no. 9, pp. 1159–1184, Oct. 2016, doi: 10.1002/spe.2442.

[38] WikiPedia. *Domain-Driven Design*. Accessed: Dec. 18, 2019. [Online]. Available: https://en.wikipedia.org/wiki/Domain-driven_design

[39] E. Evans, *Domain-Driven Design Reference: Definitions and Patterns Summaries*. Indianapolis, IN, USA: Dog Ear, 2014.

[40] E. Evans. *DDD and Microservices: At Last, Some Boundaries*. Accessed: Dec. 18, 2019. [Online]. Available: https://www.infoq.com/presentations/ddd-microservices-2016/

[41] M. Wasson. *Building Microservices on Azure*. Microsoft Docs. Accessed: Dec. 30, 2019. [Online]. Available: https://docs.microsoft.com/en-us/azure/architecture/microservices/

[42] D. Lebrero. *The LOC is Often Used to Measure Complexity, and it is Often Right. Not for C. . .—DEV*. Accessed: Dec. 18, 2019. [Online]. Available: https://dev.to/bousquetn/comment/b4o

[43] J. Lotz, A. Vogelsang, O. Benderius, and C. Berger, "Microservice architectures for advanced driver assistance systems: A case-study," in *Proc. IEEE Int. Conf. Softw. Archit. Companion (ICSA-C)*, Hamburg, Germany, Mar. 2019, pp. 45–52.

[44] F. Rademacher, S. Sachweh, and A. Zündorf, "Towards a UML profile for domain-driven design of microservice architectures," in *Software Engineering and Formal Methods*, A. Cerone and M. Roveri, Eds. Cham, Switzerland: Springer, 2018, pp. 230–245.

[45] T. Engel, M. Langermeier, B. Bauer, and A. Hofmann, "Evaluation of microservice architectures: A metric and tool-based approach," in *Information Systems in the Big Data Era*, J. Mendling and H. Mouratidis, Eds. Cham, Switzerland: Springer, 2018, pp. 74–89.

[46] Microsoft Corp. *GitHub—DotNet-Architecture/eShopOnWeb: Sample ASP.NET Core 3.1 Reference Application, Powered by Microsoft, Demonstrating a Layered Application Architecture With Monolithic Deployment Model*. Accessed: Oct. 11, 2020. [Online]. Available: https://github.com/dotnet-architecture/eShopOnWeb

[47] S. Daya, *Microservices From Theory to Practice Creating Applications in IBM Bluemix Using the Microservices Approach*. Poughkeepsie, NY, USA: IBM Corporation, 2016.

[48] F. Rademacher, J. Sorgalla, and S. Sachweh, "Challenges of domain-driven microservice design: A model-driven perspective," *IEEE Softw.*, vol. 35, no. 3, pp. 36–43, May 2018, doi: 10.1109/MS.2018.2141028.

[49] C. de la Torre, B. Wagner, and M. Rousos. *NET Microservices Architecture for Containerized NET Applications*. Accesssed: Oct. 30, 2020. [Online]. Available: https://aka.ms/microservicesebook

[50] C. S. Atole and K. V. Kale, "Assessment of package cohesion and coupling principles for predicting the quality of object oriented design," in *Proc. 1st Int. Conf. Digit. Inf. Manage.*, Bengaluru, India, Apr. 2007, pp. 1–5.

[51] Z. Xiao, I. Wijegunaratne, and X. Qiang, "Reflections on SOA and microservices," in *Proc. 4th Int. Conf. Enterprise Syst. (ES)*, Melbourne, VIC, Australia, Nov. 2016, pp. 60–67.

[52] F. Melo. *eShopOnContainers*. Accessed: Oct. 13, 2019. [Online]. Available: https://github.com/dotnet-architecture/eShopOnContainers

[53] H. Vural. *Hulyav/eShopContainersSmall*. Accessed: Oct. 30, 2020. [Online]. Available: https://github.com/hulyav/eShopContainersSmall

[54] H. Vural. *Hulyav/eShopContainersBig*. Accessed: Oct. 30, 2020. [Online]. Available: https://github.com/hulyav/eShopContainersBig

[55] Zen Program Limited. *NDepend Product Features*. Accessed: Oct. 10, 2029. [Online]. Available: https://www.ndepend.com/features/

[56] T. L. Saaty, *Decision Making for Leaders: The Analytical Hierarchy Process for Decisions in a Complex World*. Belmont, CA, USA: Lifetime Learning Publications, 1982.

[57] J. Han and M. K. J. Pei, *Data Mining: Concepts and Techniques* (The Morgan Kaufmann Series in Data Management Systems), 2nd ed. Boston, MA, USA: Morgan Kaufmann, 2006.

[58] P. Josh and G. Adam, *Deep Learning: A Practitioner's Approach*. Sebastopol, CA, USA: O'Reilly, 2017.

[59] B. Mirkin, *Clustering for Data Mining: A Data Recovery Approach* (Chapman & Hall/CRC Computer Science & Data Analysis Book 3). Boca Raton, FL, USA: Chapman & Hall/CRC, 2005.

[60] K. Teknomo. *Multi Criteria Decision Making Calculator*. Accessed: Jan. 17, 2020. [Online]. Available: https://people.revoledu.com/kardi/tutorial/AHP/MCDM_Calculator.html

[61] O. Zimmermann, "Microservices tenets," *Comput. Sci.-Res. Develop.*, vol. 32, nos. 3–4, pp. 301–310, Nov. 2016, doi: 10.1007/s00450-016-0337-0.

[62] F. Melo. *Messaging Patterns for Event-Driven Microservices*. Accessed: Oct. 14, 2019. [Online]. Available: https://content.pivotal.io/blog/messaging-patterns-for-event-driven-microservices

HULYA VURAL (Member, IEEE) received the B.S. and M.S. degrees in computer engineering from the Middle East Technical University, and the Ph.D. degree in software engineering from Atilim University, Turkey. She held a Research Assistant position and then worked on a joint Havelsan and Boeing project. She worked as a Lead Software Development Engineer at Microsoft, Seattle, WA, USA, for nine years. She is currently working as a Development Manager at Havelsan, setting up the military and government cloud for Turkey.

MURAT KOYUNCU received the Ph.D. degree in computer engineering from the Middle East Technical University, Ankara, Turkey, in 2001. He is currently a Professor with the Department of Information Systems Engineering, Atilim University, Ankara. His research interests include fuzzy logic, object-oriented databases, knowledge-based systems, multimedia databases, computer networks, and multimedia wireless sensor networks.