A Probabilistic Approach For Obtaining An Optimized Number Of Services Using Weighted Matrix And Multidimensional Scaling

Anika Sayara
Dept. of CSE
AUST
Dhaka, Bangladesh
sayara.anika@gmail.com

Md. Shamim Towhid
Dept. of CSE
Uttara University
Dhaka, Bangladesh
shamim.towhid@gmail.com

Md. Shahriar Hossain
Software Development
Insightin Technology
Dhaka, Bangladesh
shahriar_cse@live.com

***Abstract*—While migrating from Monolith to Microservices architectural style, it is often confusing to define service boundaries. This paper describes a probabilistic technique to define the services on a Monolithic system considering three major aspects - update rate, scaling rate, and technology diversity necessary to realize each service properly. To do so, we have used the weighted matrix and Multidimensional Scaling (MDS). This technique helps to define an optimal number of services for implementing the system in the Microservices Architecture. This paper will be useful for organizations trying to migrate from Monolithic System to Microservices.**

Keywords—Microservices Architecture; Monolithic System; Weighted Matrix; Multidimensional Scaling.

I. INTRODUCTION

In Monolithic Architectural Style, updating any part requires testing and redeploying the entire solution. Scaling is also not possible without scaling the entire solution. And even though different parts of the same solution may have conflicting resource requirements, in Monolithic Architecture one technology stack has to be used throughout the entire solution. Though SOA solves these problems to some extent, the shared Service Contracts, shared Databases, etc., still causes dependency problems. Microservices, on the other hand, does not support any shared spaces, each service is completely independent, and hence the Microservices Architectural style is gaining more and more popularity with passing times.

In Microservices Architecture a system is implemented as some small services that are independently deployable and scalable. However, the number and size of the services are not very clear.

The most common notion of the size of a service is that a service should be able to do one thing properly. Though it seems to correctly define the size, it does not. Because the notion of 'one thing' is not very clear. For example, in a system, if payment is one of the business capabilities then payment can be considered as one thing. But then again it is possible to split the payment into user payment and author payment, each of which can be 'one thing' individually and hence one service each. Furthermore, user payment may again be split into other smaller units each of which can be defined as a service. Hence it is unclear whether payment as a whole should be a service or should there be two services namely user payment and author payment. Thus, it is not clear to what extent should the business capabilities be split to define a service.

In this paper, we focus on solving the problem of defining services while migrating from Monolithic to Microservices architecture. We propose a generalized technique that defines services around business capabilities using multidimensional scaling and weighted matrix obtained from the probabilistic values of update rate and scaling rate of the Monolithic system. The paper is organized as follows. First, related works are presented in section II. Then our proposed method is presented in section III, which is followed by limitations of the proposed technique in section IV. Finally, the paper is concluded with some possible future directions in section V.

II. RELATED WORKS

Netflix [1], Sound cloud [2], ADP Innovation Labs [3] and many other big names have already successfully refactored their huge monolith to Microservices. In [4] Namiot et al. presented an overview of Microservices architecture. However, they did not evaluate the concepts on a real-world system. In order to modularize the system to Microservices ADP went through a process called monolith strangulation, breaking down the bureaucracy of the architecture. They identified the business capabilities and went to the code level of the application [3]. NGiNX in [5] proposed three strategies namely Stop Digging, Split Frontend and Backend and Extract Services. In Stop Digging approach new functionality is implemented by not adding code to the Monolith. Hence this approach prevents the existing Monolith to become larger but does not break down the existing Monolith to achieve Microservices. The Split Frontend and Backend approach shrinks the monolithic application by splitting the presentation layer from the business logic and data access layers but some or all of

978-1-5386-1150-0/17/$31.00 ©2017 IEEE

these layers may still be huge Monoliths. The extract services approach does not define any method to turn the existing modules to standalone services. Richardson [6] made use of use case to decompose a Monolith into subsystems while evaluating Microservices architecture. But use case often fails to describe a system. Even though Sarkar et al. implemented modularization on a Monolithic Banking system, he did not consider Microservices architecture. Terra et al. [8] proposed a technique to modularize the system through code isolation. But they do not target the extraction of independent modules, which is a distinguishing feature of Microservice-based architectures.

III. Proposed Method for Defining Optimal Number of Services

In order to define the size of a service, we suggest that –

"A service should be able to do one homogeneous thing properly. The homogeneity of each service is determined on the basis of update rate, scaling rate and technology used."

In the following sections we describe our proposed method by applying it on an example system.

A. Symbols used in the proposed technique and their meanings

- L_n: list of business capabilities.
- T_n: list of sub-business capabilities obtained by further breaking down each business capability.
- M: an m x m matrix where M_{AiAj} gives the probability of A_i and A_j being updated together given independent update probabilities of Ai and Aj respectively. (where m is the number of items in T_n and A_i, $A_j \in T_n$ and $i \neq j$).

  i.e. $M_{AiAj} = A_{i(updateprobability)} \times A_{j(updateprobability)}$ (1)

  The independent update probability of any sub-business capability, $A_x \in T_n$ is obtained by finding out how many times the sub-business capability, A_x is updated in 100 times that the business capability, $A \in L_n$ is updated.
- S: an m x m matrix obtained from update matrix M where S_{AiAj} is obtained by adding to M_{AiAj} the probability of both A_i and A_j being scaled up together and scaled down together and subtracting the probability of A_i being scaled up and A_j being scaled down at the same time and vice versa. (where m is the number of items in T_n and A_i, $A_j \in T_n$ and $i \neq j$).

  i.e.

  $$S_{AiAj} = M_{AiAj} + (A_{i(scaleupprobability)} \times A_{j(scaleupprobability)}) + (A_{i(scaledownprobability)} \times A_{j(scaledownprobability)}) - (A_{i(scaleupprobability)} \times A_{j(scaledownprobability)}) - (A_{i(scaledownprobability)} \times A_{j(scaleupprobability)})$$ (2)

  The independent scale up and independent scale down probabilities of the sub business capabilities are obtained in the same way as independent update probability.
- G_n: List of similarly weighted sub business capabilities $C_1, C_2, \ldots C_n$ obtained by applying multidimensional scaling on the weighted matrix S.
- OtherTech: an n x n matrix (where n is the number of items in G_n) having –

  $$OtherTechcicj = \begin{cases} 1, & CiCj \text{ uses similar technology} \\ 0, & CiCj \text{ uses dissimilar technology} \end{cases}$$

  Here $C_i, C_j \in G_n$
- F_n: final list of required services.

B. Algorithm of the proposed Technique

The proposed Technique uses GETSERVICES() function (Algorithm 1) to obtain optimal number of services. This function breaks down business capabilities listed in L_n into sub business capabilities $A_1, A_2 \ldots A_n$ and lists in T_n. Then Update Matrix, M is obtained by using ConstructUpdateMatrix() function (Algorithm 2) and Scaling Matrix, S is obtained using ConstructScalingMatrix() function (Algorithm 3). Finally, the list of services, F_n is obtained by calling ResolveTechConflict() function (Algorithm 4).

Algorithm 1 Algorithm for obtaining an optimal number of services

```
1. function GETSERVICES() return List of services
2.       Ln = A, B…N    where A, B…N are business
                        capabilities
3.      repeat
4.          for each itemx in Ln do
5.                Tn = A1, A2...An obtained by breaking
                       down itemx
6.                M = ConstructUpdateMatrix(List Tn)
7.                S = ConstructScalingMatrix(Matrix M,
                                        List Tn)
8.                Fn ← ResolveTechConflict(Matrix S)
9.          end for
10.     until the monolith disappears entirely or becomes
        small enough that it is just another service
11.     return Fn
12. end function
```

ConstructUpdateMatrix() (Algorithm 2) uses equation (1) to obtain Update Matrix, M.

ConstructScalingMatrix() (Algorithm 3) uses equation (2) to obtain Scaling Matrix, S.

The ResolveTechConflict() function groups together the sub-business capabilities having similar weight by applying

multidimensional scaling on the weighted matrix S. After grouping the equally weighted items together, technology to be used for their implementation is considered. If any item in a group is seen to be better implemented using a different technology than the other items in the group, then it is separated from that group and defined as a separate service. Otherwise all similarly weighted items having similar technology are defined as a service. For example, let A_1, A_2 and A_3 are grouped together by MDS and let all of them are implemented using a relational database. If it is seen that A_2 can have better implementation using NoSQL then A_1 and A_3 together will be a service and A_2 will be a separate service.

Algorithm 2 Algorithm for constructing update matrix

```
1.  function ConstructUpdateMatrix(List Tn) return
    Matrix M
2.      for each itemx in Tn do
3.          Find independent update probability of itemx
4.      end for
5.      for i=0 to i < length(Tn) do
6.          for j=0 to j < length(Tn) do
7.              if i==j then
8.                  MAiAj = 0
9.              else
10.                 MAiAj = Ai(updateprobability) x Aj(updateprobability)
11.             end if
12.         end for
13.     end for
14. return M
15. end function
```

Algorithm 3 Algorithm for constructing scaling matrix

```
1.  function ConstructScalingMatrix(Matrix M, List Tn)
    return Matrix S
2.      for each itemx in Tn do
3.          Find independent scale up probability of itemx
4.          Find independent scale down probability of
            itemx
5.      end for
6.      for i=0 to i < length(Tn) do
7.          for j=0 to j < length(Tn) do
8.              if i==j then
9.                  SAiAj = 0
10.             else
11.                 SAiAj = MAiAj +
                        (Ai(scaleupprobability) x Aj(scaleupprobability)) +
                        (Ai(scaledownprobability)xAj(scaledownprobability)) -
                        (Ai(scaleupprobability)xAj(scaledownprobability))-
                        (Ai(scaledownprobability) x Aj(scaleupprobability))
12.             end if
13.         end for
14.     end for
15. return S
16. end function
```

Algorithm 4 Algorithm for resolving technology conflict

```
1.  function ResolveTechConflict(Matrix S) return List
    Un
2.      using MDS group almost equally weighted items
        of S in Gn
3.      Un ← {}
4.      for each group Gn obtained by MDS do
5.          if Cn is the only item in Gn then
6.              Un ← Un ∪ Cn
7.          else
8.              Create array Pt of size equal to length(Gn)
9.              for i=0 to i<length(Pt)
10.                 Pt[i] = 0
11.             end for
12.             for i=0 to i < length(Gn) do
13.               for j=i+1 to j < length(Gn) do
14.                 if Ci and Cj uses same tech then
15.                     OtherTech[i][j] = 1
16.                 else
17.                     OtherTech[i][j] = 0
18.                 end if
19.               end for
20.             end for
21.             k=0
22.             flag = 0
23.             for i=0 to i < length(Gn) do
24.               for j=i+1 to j < length(Gn) do
25.                 if Pt[i] = 0 then
26.                   if OtherTech[i][j] = 1 then
27.                     Xk ← Xk ∪ Cj
28.                     Pt[j] = 1
29.                     flag = 1
30.                   end if
31.                 end if
32.               end for
33.               if flag = 0 and Pt[i] = 0 then
34.                 Un ← Un ∪ Ci
35.               else if flag = 1 and Pt[i] = 0 then
36.                 Xk ← Xk ∪ Ci
37.                 Un ← Un ∪ Xk
38.                 Pt[i] = 1
39.                 k = k + 1
40.                 flag = 0
41.               end if
42.             end for
43.         end if
44.     end for
45.     return Un
46. end function
```

C. *Explanation of the proposed algorithm with an example*

Consider an application where users can browse books submitted by the subscribed authors, download the books as pdf, give feedback in the form of ratings and reviews and join different book clubs. There is a recommendation system that

suggests books to users based on their feedbacks and type of book clubs they joined. Also, third party search based recommendation is provided along with region based recommendation for absolutely new users and two types of user subscriptions - monthly and one time. Authors are directly added by the admin and they can submit their stories which are later verified by the admin. If published, the author gets a fixed payment.

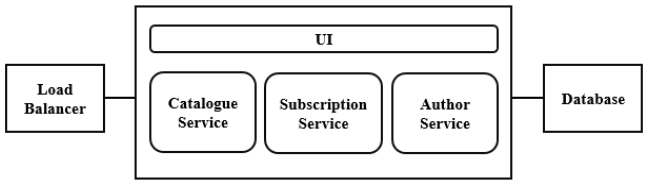


Fig. 1. Example application in monolithic architecture

The example application when implemented in monolithic architecture is shown in figure 1. Now according to algorithm, the first task is to identify the broad business capabilities. From the figure we see that broadly the application has the following three business capabilities -

- **Catalogue Service:** This service is responsible for displaying book list to the user using recommendation system. New users are shown book list using region based recommendation system and old users are shown book list using user history, that is - user feedback, joined book clubs and third-party search based recommendation system. This service also includes browsing system, that is the service is responsible for allowing users to browse books through search and finally the service also includes downloading service that allows users to download books as pdf from the book list.
- **Subscription Service:** The subscription service is responsible for granting new subscribers, managing fixed payment as well as monthly payment, terminating subscription of users who are unable to pay monthly fees and who want to terminate subscription by choice. Login and Account management also falls under this service.
- **Author Service:** Author service is responsible for adding new authors, managing author profile, managing author payment, accepting submissions from the authors, granting submissions.

Thus, according to algorithm –

L_n = {CatalogueService, SubscriptionService, AuthorService}

Now for the first iteration, $item_x$ = CatalogueService $\in L_n$

Thus by breaking down CatalogueService into sub-business capabilities (as shown in figure 3) using the description of the service we have -

- A ← RegionalRecommendationService
- B ← UserFeedbackBasedRecommendationService
- C ← JoinedBookClubsBasedRecommendationService
- D ← ThirdPartySearchBasedRecommendationService
- E ← BrowsingService
- F ← DownloadingService

Thus, T_n = {A, B, C, D, E, F}

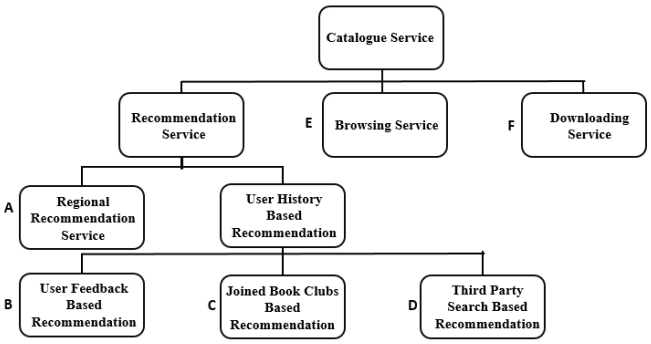


Fig. 2. Splitting the business capability 'Catalogue Service' into sub business capabilities - A, B, C, D, E and F

Now let the independent update probabilities for the A, B, C, D, E and F be –

$A_{updateprobability} \leftarrow 15/100 = 0.15$

$B_{updateprobability} \leftarrow 20/100 = 0.20$

$C_{updateprobability} \leftarrow 20/100 = 0.20$

$D_{updateprobability} \leftarrow 40/100 = 0.40$

$E_{updateprobability} \leftarrow 3/100 = 0.03$

$F_{updateprobability} \leftarrow 2/100 = 0.02$

Hence, we construct the update matrix M using (1) –

$$M_{AB} = A_{updateprobability} \text{ x } B_{updateprobability} = 0.15 \times 0.20 = 0.03$$

Here the diagonal elements are not needed since they represent independent update probability of same service and hence we make them 0.

Thus, the Update Matrix that we obtain is as follows –

$$M = \begin{array}{c} \begin{matrix} A & B & C & D & E & F \end{matrix} \\ \begin{bmatrix} 0 & 0.03 & 0.03 & 0.06 & 0.0045 & 0.003 \\ 0.03 & 0 & 0.04 & 0.08 & 0.006 & 0.004 \\ 0.03 & 0.04 & 0 & 0.08 & 0.006 & 0.004 \\ 0.06 & 0.08 & 0.08 & 0 & 0.012 & 0.008 \\ 0.0045 & 0.006 & 0.006 & 0.012 & 0 & 0.0006 \\ 0.003 & 0.004 & 0.004 & 0.008 & 0.0006 & 0 \end{bmatrix} \end{array}$$

The next step is to modify the update matrix in order to get the scaling matrix, which is nothing but a weighted matrix obtained after considering the scaling parameter. To do this we need the independent scale up and scale down probabilities of each of the services. Let the independent scale up probabilities for the services be –

$A_{scaleupprobability} \leftarrow 75/100 = 0.75$

$B_{scaleupprobability} \leftarrow 80/100 = 0.80$

$C_{scaleupprobability} \leftarrow 82/100 = 0.82$

$D_{scaleupprobability} \leftarrow 65/100 = 0.65$

$E_{scaleupprobability} \leftarrow 87/100 = 0.87$

$F_{scaleupprobability} \leftarrow 78/100 = 0.78$

Then the independent scale down probabilities for the services is given by –

$A_{scaledownprobability} \leftarrow 1 - A_{scaleupprobability} = 1 - 0.75 = 0.25$
$B_{scaledownprobability} \leftarrow 1 - B_{scaleupprobability} = 1 - 0.80 = 0.20$
$C_{scaledownprobability} \leftarrow 1 - C_{scaleupprobability} = 1 - 0.82 = 0.18$
$D_{scaledownprobability} \leftarrow 1 - D_{scaleupprobability} = 1 - 0.65 = 0.35$
$E_{scaledownprobability} \leftarrow 1 - E_{scaleupprobability} = 1 - 0.87 = 0.13$
$F_{scaledownprobability} \leftarrow 1 - F_{scaleupprobability} = 1 - 0.78 = 0.22$

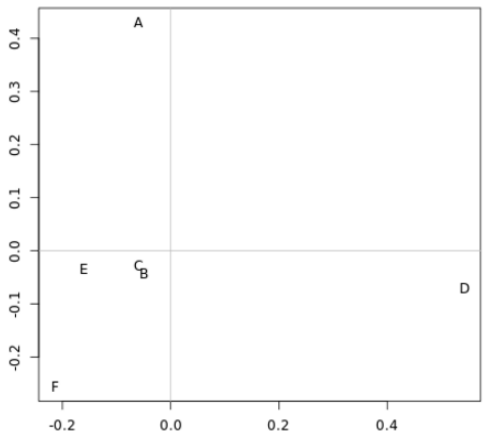
In scaling matrix S, S_{AB} is the weight assigned to the pair of services A and B. And is obtained by Equation (2) as –

$$S_{AB} = S_{BA}$$
$$= 0.03 + (0.75 \times 0.80) + (0.25 \times 0.20) - \{(0.75 \times 0.20) + (0.25 \times 0.80)\} = 0.33$$

Since the diagonal values are not needed we make them 0.

$$S = \begin{array}{c} \begin{matrix} A & B & C & D & E & F \end{matrix} \\ \begin{bmatrix} 0 & 0.33 & 0.35 & 0.21 & 0.37 & 0.28 \\ 0.33 & 0 & 0.42 & 0.26 & 0.45 & 0.34 \\ 0.35 & 0.42 & 0 & 0.27 & 0.48 & 0.36 \\ 0.21 & 0.26 & 0.27 & 0 & 0.23 & 0.18 \\ 0.37 & 0.45 & 0.48 & 0.23 & 0 & 0.41 \\ 0.28 & 0.34 & 0.36 & 0.18 & 0.41 & 0 \end{bmatrix} \end{array}$$

Now we apply MDS (Multidimensional Scaling Technique) on S (Figure 3) in order to group the similarly weighted elements



of the matrix together.

Fig. 3. Graph obtained by applying the MDS technique on matrix S

From figure 3 we see that B and C are almost equally weighted in terms of update and scalability. Now for the final step we need to consider technology conflict. The term technology conflict means that a service can be implemented more appropriately using a different technology than the one that was used to implement it in the monolithic. Technology conflict arises in monoliths because in monoliths we need to consider tradeoffs between services since the whole application uses same type of technology. For example, if the application is in .NET, the entire monolith is in .NET, however Microservices overcomes this issue and allows using different technologies for different services. Thus, while deciding on the service boundary, this should be an important consideration. Any service that requires a different technology must be declared as a separate service even though they are almost equally weighted with respect to other parameters.

Now according to the algorithm, we first group the equally weighted items together using figure 3.

$G1 \leftarrow A$

$G2 \leftarrow B, C$

$G3 \leftarrow D$

$G4 \leftarrow E$

$G5 \leftarrow F$

Since G_1, G_3, G_4 and G_5 each has only one item, each of those items can be individually defined as service. However, in case of G_2 we obtain OtherTech Matrix using Algorithm 4 as follows –

$$\text{OtherTech} = \begin{bmatrix} x & 1 \\ x & x \end{bmatrix}$$

Since OtherTech[0][1] = 1, $X_1 \leftarrow C_0, C_1$ (where C_0 = B and C_1 = C). Thus we have,

$$X \leftarrow B, C$$

$$U_n \leftarrow A, X, D, E, F$$

$$F_n = U_n$$

Thus, we have, Fn ← A, X, D, E, F [where X is B and C]

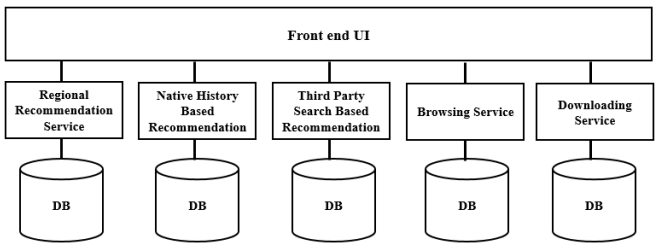


Figure 4: CatalogueService in Microservices Architecture implemented using the service boundaries obtained by applying the proposed technique.

Therefore, the obtained services are:

- Service 1: Regional Recommendation Service.
- Service 2: User Feedback Based Recommendation and Joined Book Clubs based Recommendation. These two together are renamed as Native History Based Recommendation Service.

- Service 3: Third Party Search Based Recommendation Service.
- Service 4: Browsing Service.
- Service 5: Downloading Service.

D. *Reasons for choosing update rate, scaling rate and technology diversity as parameters for obtaining service boundaries*

If we look closely at the reasons for migrating to microservices in the first place, these three are the most significant. Apart from making the entire solution more comprehensive and more understandable, Microservices architecture aims to make sure that instead of scaling up the entire solution, only the service requiring greater resource is scaled up and thus reducing the cost overhead of scaling up the other portion of the application that does not require greater resources. Other than this Microservices aims at ensuring that only the portion of the application that requires updating goes through redeployment process and if any bug occurs only that portion is affected. And finally, technology lock-ins and technology tradeoffs are eliminated in Microservices. Since these are the achievable factors while migrating to Microservices, we have considered these three as parameters for assigning weight to the sub-business capabilities. And then services are obtained by keeping the similarly weighted sub-business capabilities using same technology together, while keeping the dissimilar weighted sub-business capabilities or having different technology separately.

E. *Reason for choosing MDS algorithm*

There are many clustering algorithms i.e. K means, UPGMA etc. But among them we chose here MDS because of some reasons. In our proposed algorithm we used MDS to cluster/group the similar services into one group. Generally what MDS do is it projects the value in an N dimensional space which is specified. Then simply it calculates the distances or similarities. For our case we need to group services according to similarities based on weighted matrix. This weighted matrix comes from the probabilistic value of update or scale each business capabilities identified in the first steps. MDS was chosen instead of K means or UPGMA because k-means does not support distance matrices. It never uses point-to-point distances. Again, UPGMA is used for unweighted matrix clustering but here we have a weighted matrix. So MDS is the best option for us.

IV. LIMITATIONS OF THE PROPOSED TECHNIQUE

The limitations of the proposed technique for defining the services are listed as follows:

- The technique does not clearly define the splitting of the business capabilities into sub-business capabilities. However, the splitting can be done using business case diagram or the planning of application.
- The technique is heavily dependent on probabilities of a sub-business capability being updated or scaled. Hence the technique cannot be applied at the initial phase of application development. Also, these probability values are crucial and if for some reason, these probabilities are overestimated or underestimated, the service boundaries won't be correctly obtained.
- The technique may group together two entirely different types of sub-business capabilities if their weights become almost equal.
- The technique only defines the service boundaries or what sub-business capabilities will make up a service, it does not deal with actually splitting the monolith by eliminating dependencies and breaking up the back-end database, etc.

V. CONCLUSION

In this paper, we have shown a technique for obtaining services from an existing Monolith. To do so we used weighted matrix obtained from update and scale probability values. Then we applied Multidimensional Scaling to group together the similarly weighted sub business capabilities together. Next, the items that could be better implemented using same technology were grouped together and defined as a separate service.

REFERENCES

[1] Adrian Cockroft (2014). Migrating to Microservices by Adrian Cockroft (Part 1) [online]. Available: https://www.youtube.com/watch?v=1wiMLkXz26M

[2] Phil Calcado. (2014). Building Products at SoundCloud—Part II-Breaking the Monolith [online]. Available: https://developers.soundcloud.com/blog/building-products-at-soundcloud-part-2-breaking-the-monolith

[3] Jason Melo. (2015). Microservices Day: Monolith to Microservices [online]. Available: https://www.youtube.com/watch?v=Jv9HFFwnjqA

[4] Dmitry Namiot and Manfred Sneps-Sneppe. On micro-services architecture. International Journal of Open Information Technologies, 2(9):24–27, 2014.

[5] Chris Richardson. (2016). Refactoring a Monolith into Microservices [online]. Available: http://www.nginx.com/blog/refactoring-a-monolith-into-microservices

[6] Chris Richardson. (2014). Microservices: Decomposing applications for deployability and scalability [online]. Available: http://www.infoq.com/articles/microservices-intro

[7] Santonu Sarkar, Shubha Ramachandran, G. Sathish Kumar, Madhu K. Iyengar, K. Rangarajan, and Saravanan Sivagnanam. Modularization of a large-scale business application: A case study. IEEE Software, published by IEEE Computer Society, 26:28–35, 2009.

[8] Ricardo Terra, Marco Tulio Valente, and Roberto S. Bigonha. An approach for extracting modules from monolithic software architectures. IX Workshop de Manutenção de Software Moderna (WMSWM), pages 1–18, 2012