

The Adoption of Microservices Architecture as a Natural Consequence of Legacy System Migration at Police Intelligence Department

Murilo Góes de Almeida(✉) and Edna Dias Canedo(✉)

Department of Computer Science, University of Brasília (UnB),
P.O. Box 4466, Brasília–DF, Brazil
murilow@gmail.com, ednacanedo@unb.br
http://ppca.unb.br/

Abstract. During the end of 2017, the systems developed by policemen in the Intelligence Department of São Paulo State Military Police were outdated in technology and did not follow any kind of pattern, neither had integration with services, databases and other systems, being very difficult to maintain. However, this kind of system could not be abandoned, because they were important in the organization. Thus, the managers of the Intelligence Department understood the need for modernization of architecture and system technologies. There are some approaches that can be chosen in this evolution process. A difficult question to be answered is related to which path is better to be followed in this process of system migration, mainly about the choosing of monolithic or microservices architecture. Although the microservice architecture is modern, it is not so easy to be implemented directly in a legacy system evolution process, because it depends on a strong domain of businesses areas and technology, which makes it difficult to involve of beginners developers. In this paper, we reported an experience about migrating a legacy system used in Intelligence Department of Sao Paulo State Military Police to a microservice-based architecture by re-engineering the legacy system as a monolith first and then decomposing it to microservices. We realized that microservices did not necessarily had to be adopted in the first place, but it could be a natural consequence of evolution, using a monolithic approach in the beginning. This process may also help the development team to improve their knowledge about the adopting of new technologies, architectures and better understanding of business.

Keywords: Monolithic · Microservice · Legacy system · Architecture migration · Software design tradeoffs

1 Introduction

Public security in Brazil is a state duty, guaranteed by Federal Constitution, that means, the public power is responsible to preserve the public order, security of

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2022
O. Gervasi et al. (Eds.): ICCSA 2022, LNCS 13375, pp. 354–369, 2022.
https://doi.org/10.1007/978-3-031-10522-7_25

people and property. There are law enforcement institutions created by Federal Constitution [7], each one having a specific duty in the public security context. Brazil is a Federal Republic, divided by 26 states plus a Federal District. The states have their own public security system, responsible for prevention and repression of crimes.

The organization responsible for preventing crimes is named Military Police. It has an intelligence department, that works for security public intelligence activities in São Paulo State and part of the brazilian public security intelligence subsystem [8]. This department develops its own systems, as also manages technology infrastructure which provides the systems sustentation. These activities are all developed by military police people, because of data sensitivity which is processed and stored, most of them classified. Thus, the intelligence department does not have an outsourced technical staff hired to work with technology, using policemen able to work in intelligence activity. Furthermore, it is very tricky to find a policeman with technology skills, because it is not required in the police admission process. Therefore, the policeman staff working with technology in intelligence is very reduced compared to other institutions, moreover, they do not have advanced knowledge in system development, although they have a great desire to learn it.

Because of the small number of police officers, added with limited knowledge in technology, at the end of 2017, systems developed by policemen in the Intelligence Department did not follow any kind of pattern, neither had integration with services or databases, thus, there were a lot of legacy monolithic systems that did not "talk" with each other. Each system had its own authentication system, frontend, backend and database.

The software development process used at that moment involved the use of a "scaffold" tool, that generates automatically all system code using a model first approach, that means, having a previously database modeled using some visual tool. Scaffold may be a good solution for programming learning, it is encouraged for novice programmers who have limited knowledge and experience [29]. The using of this process was justified because there was a reduced staff added with the client expectation of receiving a finished system in a short deadline.

In fact, the process mentioned above accelerated the process of development and delivery of systems, however, it did not accomplish the user's expectations. The system maintenance was very complicated, because the code generated by the scaffold tool was very difficult to read and edit. The most worrying was the fact of the systems, even of different business areas inside the police intelligence, did not communicate between them, neither with other corporate systems of Military Police, consequently, behaving as "black-boxes", and, in the majority of cases, storing exactly the same kind of data in different places, causing several redundancies of data. Last but not least, some intelligence business areas which did not have a developed application, used to store important data in files like spreadsheets, without any kind of integration with other areas. Figure 1 shows how the systems and database were organized in the scenario above.

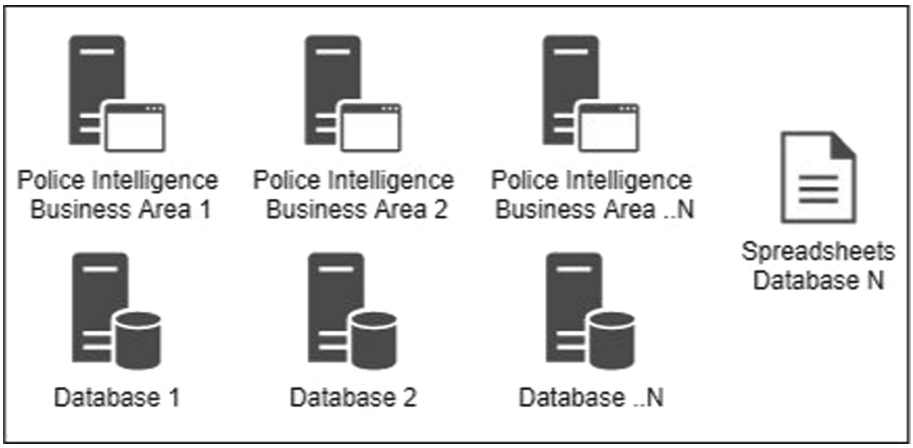

Fig. 1. Isolated and monolithic systems and databases developed in Intelligence Department

According to the presented scenario, the managers of the Intelligence Department understood the need for modernization of architecture and system technologies, assuming that the main goal had to be the data integration and systems interoperability. In this paper, we demonstrated how this process was developed, since the mapping of data by business fields, the choosing of system architecture and technologies. After that, it is discussed why we did not use microservices at first, but this adoption became a consequence of this system evolution process.

Accordingly, this paper presents the following contributions:

1. In a process of legacy systems migration, the microservices architecture is not necessarily the best option to be implemented immediately;
2. The use of a monolithic architecture combined with modern technologies helps the development team to understand the business areas that can later be transformed into microservices;
3. The need to use a microservices architecture may arise naturally when the monolithic architecture becomes unsustainable, but easy to be split into microservices;
4. The details about the migration process in a context of public service in which the development team is small and with little experience about this subject;

2 Background

2.1 Legacy Systems

Legacy systems, in the words of Bennett [3], can be defined informally as "large software systems that we don't know how to cope with but that are vital to our organization". Generally, the legacy systems become very difficult to maintain

and evolve over time, however, the majority of them perform crucial work for their organization. Consequently, at a given time, legacy systems cost a lot of efforts, causing a paradox, although they are vital for the organization, at the same time are causing several problems for them. Therefore, there are several studies about legacy systems migration.

Bisbal et al. [4] suggested migration methodologies, no matter what will be the new technology or architecture. There is a methodology more aggressive where the entire legacy system is abandoned and redeveloped from the scratch, nevertheless, it is also possible to migrate taking advantage of what already exists. There are also studies about migration to specifics architectures, such as migrating legacy systems towards object-oriented platform [14] or even migration to more modern architectures such as microservices [23]. Indeed, migration of legacy systems is a challenge that can be faced by almost every organization in the world, what is demonstrated in this specific case of our study.

2.2 Services Oriented Architecture

The term SOA, abbreviation to Service Oriented Architecture, was first mentioned in the industry by Gartner Group in 1996 [1]. There are several definitions about Service Oriented Architecture, nevertheless, all of them agree that it is a paradigm which improves flexibility in systems [22]. An adequate SOA definition is a software architecture based on services, which are loosely coupled software components that can be orchestrated to increase business agility [17]. This concept helps us to understand that Services Oriented Architecture brings some advantages, such as reliability, scalability, reusability, loosely coupled, easy maintenance, among others. The understanding about SOA helps our comprehension about microservices architecture, which has a relationship with SOA concepts.

2.3 Monolithic Architecture

Monolithic Architecture is a traditional strategy to software development. In this architecture, the functions are encapsulated in the same application [13]. If the monolith application is simple, it is advantageous, because it is easy to develop, test and deploy [13]. Otherwise, it may be more complicated to manage in some scenarios, because a monolithic architecture shares the physical resources of the same machine, such as memory, databases or files. A very noted problem in this architecture is related to scalability and the aspects related to change and evolution of application [9].

In other words, if a monolithic application has a lot of functions, it is not possible to manage individually the amount of physical resources that each function has to consume. The entire resources will be provided to the monolith, not exclusively to functions. Finally, if some functionality has to be repaired or evolved, all application will have to be affected, because it is about the entire application, no matter what is the function.

2.4 Microservices Architecture

Microservices are small scalable applications that represent bounded business capabilities and is frequently deployed independently [9] and are mentioned in the bibliography as architecture and paradigm. Using this approach, each microservice works like an independent application representing a single business functionality. It helps in terms of scalability. Independent scalability is very different comparing to monolith, because in this case, when a microservice consumes more resources, it is easy to provide more physical resources especially for that application. There is also a pecuniary advantage in this case, because it is possible to scale the applications only when it is demanded [25].

It is important to mention that microservices also provide independent development. A single unit can be developed by a specific group, that can use a specific programming language [25]. The independent development also brings others benefits, in the words of Bucchiarone et al. [9], "Since each microservice represents a single business capability, which is delivered and updated independently, discovering bugs or adding minor improvements does not have any impact on other services and on their releases". It is a great advantage compared to monolith, which a simple maintenance or evolution could affect the entire application. The same author also points that the use of microservices can be developed and managed by a single team. In other words, it is possible to have an entire ecosystem of microservices and different groups working with each application, isolated.

The adoption of microservices architecture may brings some drawbacks and challenges, such as a more complex deployment, performance (generally microservices communicate over the network), physical resources consumption and others [27], thus, the adoption of microservices architecture will never necessarily be better than the monolithic one, because it depends on a specific case.

3 Business Field Data Mapping

In this section, we demonstrate an investigation about where the data of different business areas of Military Police of São Paulo are stored and how they can be consumed by intelligence applications.

The Military Police of São Paulo State has a specific Information Technology Department separated to Intelligence Department, also managed by police officers, having their own software factory and infrastructure, nevertheless, they can work with outsource professionals, being responsible by operational and administrative systems in the majority of business areas of police (except intelligence), such as operational, human resources, logistics, health and financial.

There was a general agreement that systems developed by the intelligence field had to store only data related to their own business areas, that means, only what is produced by the intelligence service. Data out of intelligence business domain, such as human resources, should be accessed through that system which stores this original data, using some kind of systemic integration.

The data sources from different business areas have been mapped, after that, we realized that the majority of them were under responsibility of the Information Technology Department from Military Police. There was possibility to access some data sources using web services. In this way, external data that in the past were replicated in a database of Intelligence Department, started to be accessed directly through web services. Because of that, Intelligence Department stopped storing redundant data, that used to consume unnecessary storage volume and sometimes did not have integrity.

During the mapping activity in the Information Technology Department of Military Police it was possible to figure out that there was available an authentication service that could be integrated in the intelligence applications. In this case, the majority of systems developed by that department uses an authentication service provided by Active Directory (AD). AD is a common repository for information about objects that reside on the network, such as users, among others [15]. The using of AD as authentication service would help to solve frequent complains from intelligence users and developers, because they used to forgot the password, and the intelligence systems did not have a password recovering functionality, thus, every time that some user forgot the password, they had to call or send an email to intelligence developers, that would be able to reset or change password.

We concluded that the authentication process had to be provided by AD, because the users and passwords stored in that service were used in the majority of police systems, therefore, the user would have less probability to forget it, and, even if that happened, they would have to deal directly with the Information Technology Department from Military Police, which has more staff to handle with it. Thus, the systems of intelligence stopped checking user authentication, it would be provided by AD service, however, the intelligence system had to check if a user had authorization to login and what functionalities he was able to access.

The correct understanding about what kind of data is related to intelligence business core provides a possibility to develop a system more flexible, integrated and without data redundancy, in addition to opening more discoveries of services for integration, such as the case of authentication service adoption. The result of this mapping provided a new system design integration as it is represented in Fig. 2. After this process, both development team and decision makers reinforced the understanding that system integration through a service oriented architecture helps to reduce data redundancy and keeps the data with a integrity assurance.

4 Legacy System Migration

In this section, we explained what happened during the legacy system migration process. It is pointed new routines that started to be followed, as a consequence of modernization. At the end, it is explained when the monolithic architecture started to be difficult to maintain, but easy to migrate into microservice architecture.

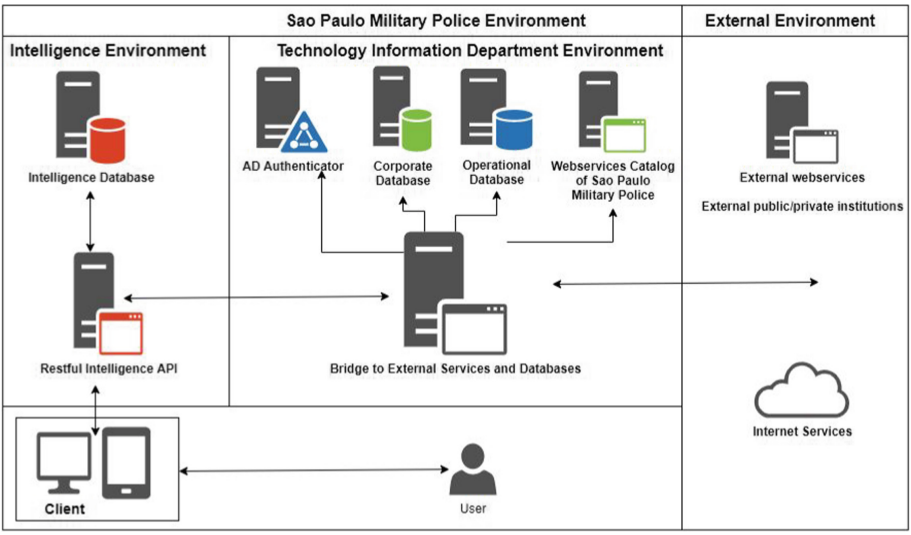

Fig. 2. New integration API proposed after a Business Field Data Mapping

After the process of business data understanding and mapping, it was proposed the building of a platform using only a frontend that concentrates all system functionalities, according to user system permissions. Different from legacy system, the new one has a total separation of frontend to backend, being this last one a RESTFUL API, able to communicate with externals web-services of different business areas. In RESTful applications, the method information goes into HTTP method [28], and implements REST (Representational State Transfer) [18], which has as key abstraction of information the resource, a conceptual mapping to a set of entities. For using the REST architecture, The use of a RESTFUL API approach seems easy to understand and implement, facilitating the integration with different clients, such as a mobile application, beyond only a frontend web application.

The using of a "scaffold" tool was abandoned, and the development team started to produce their own code, taking advantage of this moment to adopt a GIT repository, which did not existed in the system development process since now. GIT is a version control system (VCS), it allows you to track the iterative changes you make to your code [5]. The development team did not know how GIT worked, therefore, it was necessary a training, not only related to technical details, but also giving a better understanding about the importance of working with code versioning in a repository. It seems complicated at first, however, is a powerful tool that can improve code development and documentation [5].

In this first moment, we did not use a microservice architecture, because the crew had little knowledge about this subject, and also a difficult to know how to model each microservice according to application context, thus, we preferred to adopt a monolithic architecture. Although we used this approach, the source

code organization was carefully organized, especially the classes, packages and their respective names, separated by their business areas or even the functionality performed by the class. This separation contributed to the development team having a greater understanding of what could actually be broken in microservice, if the need arose, throughout the evolution of the application.

Over the months, new features were added to the system, both from new business areas that were not integrated in the platform, as well as evolution of the existing ones. This made the project, even well organized, starting to have a higher number of classes and lines of codes, in addition to taking more time to build the application in the production environment, thus, a simple change in a specific module, it affected all users, without distinction. The high number of features was also making it difficult to monitor which ones were most accessed by users or even which consumed more hardware resources. If for some reason the application was slow, it ended up being scaled in its entirety, without the possibility of identifying the real reason for the slowness.

There was no continuous integration process in the sense of building automation, that is, the building of a new functionality in the production environment was generated and replaced by the old one manually, which made the system unavailable for a certain moment. This unavailability, even if temporally, caused certain annoyances, such as users contacting the technical team and informing them that the application was not working. The manual process of updating builds also made it risky. If the responsible did it incorrectly, making the rollback process extremely difficult.

As the project started to increase the complexity in its source code, although well organized, it caused insecurity for the development team in carrying out maintenance or evolutions, especially regarding newer developers. The same insecurity was also on the part of the more experienced staff, fearing that the handing over of the project responsibilities to the beginning staff could somehow cause problems that would affect all application, and, because of that, the workload ended staying with the more experienced developers, preferring to perform actions that, in theory, the beginning staff would be able to work.

For theses reasons, the use of the microservices approach started to be considered. The expectation of the development team and decision makers was that breaking this monolith into microservices would solve the majority of these problems and maybe would not be risky, because at this point, the project was very well organized and modularized and the knowledge about the functionalities and business areas were very clear to all involved people in this project.

5 Microservices Adoption

In this section, we present which technologies were chosen to adopt microservices and the reasons, the advantages and the challenges faced during this process.

The monolithic application was being developed with the Java Spring Boot framework. According to official documentation, Spring makes it easier to create Java enterprise applications. It provides everything you need to embrace the Java language in an enterprise environment [21], and luckily, the same framework ecosystem has its own libraries for using applications as microservices, in this case, the monolithic to microservices migration process, it was used the library called "Spring Cloud". Spring Cloud is an umbrella project composed of several projects related to building cloud-based applications and microservices [12]. It provides basically, among other possibilities, a dependency named "Discovery Server" that behaves like a server where the other application subscribes to it, whenever they start up. Spring Cloud also provides another dependency that works as a "Gateway", which also subscribes to Eureka Server and checks which applications are subscribed there, and finally organizes all routes starting from a single address.

There was considerable ease in adopting the technologies mentioned above, because the development team was already acquainted with the Java programming language and the Spring framework. The separation of the project into microservices was not an obstacle either, after a certain time working with the monolithic approach, taking care to organize the code into specific packages, it helped to have a greater understanding of which of those features should present itself as an independent microservice, thereby, the monolithic application was broken into 13 (thirteen) independent microservices. Figure 3 shows a new environment using microservices in the intelligence infrastructure.

5.1 Advantages

The migration process to microservices also "forced" the development team to verify which tools would be best suited to support the applications. All microservices were hosted into Docker containers and The Windows Server operating system was replaced by a Linux distribution. The applications that are built in the docker are packaged with all the supporting dependencies into a standard form called container and brings some advantages such as speed, portability, scalability, rapid delivery, and density [6].

It also took the opportunity to resolve the issue of continuous integration. Due to the migration to microservices, there was a specific project for each microservice, therefore, using the GitFlow workflow in integration with the Jenkins Continuous Integration server, it was possible to create an automatic deploy flow in each microservice within a specific environment (development, approval and production) depending on the branch where the code was committed, such as deploy in the development environment for the branch marked as "feature", or in production for "master". Jenkins is a continuous integration tool, it can simplify and accelerate delivery by helping you automate the deployment process [30]. The implementation of it allowed us to automate the construction, testing and deployment of each microservice, greatly facilitating the continuous delivery process and GitFlow adoption, an important abstraction for flow state management which allows us to achieve higher levels of safety, provenance, ease

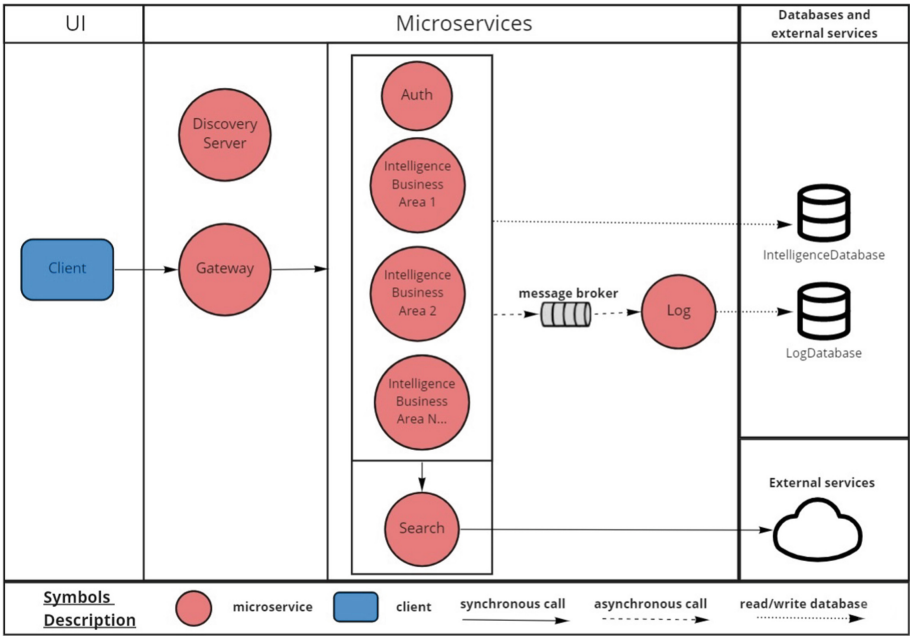

Fig. 3. Environment using microservices in intelligence side

of programmability, and support for multiple applications [16]. In addition to collaborating in this process, it enriched the development team's vision of good code versioning practices.

Breaking the monolithic application into smaller parts helped the more skilled developers to have greater confidence in making the source code available for the less skilled ones to explore it, because some of them isolated were very simple to understand and manipulate. There was a consensus that, if any incorrect code changes were made in some microservices, the impact would be minimal, as only that specific one would be affected, which actually happened, being easy to revert to the previous state. As a result, each developer became responsible for a specific application, according to their technical knowledge, increasing productivity and making the entire development team feel more involved with the project and more resolute in making their microservice best factored and evolved.

The use of microservices in containers facilitated scalability management, enabling a more rational distribution of hardware resources among them. At that time, it was evident that certain features of the monolithic application were less used compared to others, and after the separation, there was a better distribution of resources such as processing and memory in specific applications, unlike others that did not require as much computational resources.

5.2 Disadvantages and Challenges

Despite all the mentioned advantages, after the complete migration in the production environment from the monolithic architecture to microservices, some problems started to be identified. The first one happened when it was realized that a specific microservice consumed an expensive amount of memory, also affecting those who communicated with it. This difficulty evidenced slow communication with the database, which until then had not been seen in the monolithic approach, as this used a computational resource higher than what is used in each container individually. The moment was taken to improve the quality of queries performed on the database, in addition to optimizing the management of connection pools. At that time, what seemed like a difficulty, turned out to be advantageous, as the development team acquired new knowledge and overcame a challenge that would hardly be seen in the monolithic scenario, causing a positive refactoring in the source code and better communication configuration with the database.

Equally important to note is the fact that each microservice works as an independent application, therefore, it depends on a minimum amount of physical resources, especially memory. Thus, if the application has a large number of microservices, the available physical computational resources must be greater compared to the monolithic architecture. It was observed during this process because the virtual machine which hosts the monolithic application has 16GB of RAM memory comparing to 20GB in the virtual machine that runs the microservices ecosystem.

6 Related Work

Luz et al. [25] studied adoption of Microservices in Brazilian Government Institutions. They argue that there are many benefits in microservices adoption, including better scalability, productivity and maintainability. Nevertheless, it can be a challenging task in Government Institutions, mainly because it not only demands the understanding of new techniques and tools, but it also increases the need to automate tasks related to software deployment and software monitoring. They also conclude that it is difficult to break a monolithic system into microservices, mainly because there is a lack of guidelines with more precise descriptions about the expected granularity of a microservice. In fact, both advantages and disadvantages were faced in our migration process, the main difficulty was related to our development team skills and how to model the microservices, that were achieved during the modernization process.

Agilar et al. [31] studied a Service Oriented Approach to the Modernization of Legacy Systems in a Government University Institution. The use of Service Oriented Architecture (SOA) is mentioned as a way to solve the system modernization, using the Representational State Transfer (REST) architecture. In broad strokes, the using of REST is justified because of this simplicity and ease of understand, in addiction to the fact that the systems in study is able to work

with HTTP requests. The use of a Service Oriented Architecture with REST in the Intelligence Department are justified in general with the same arguments.

Carvalho et al. [10] conducted a study about extraction of configurable and reusable microservices from legacy systems, interviewing specialists who worked in migration process from monolithic to microservices architecture. The majority of interviewed specialists answered that variability as useful or very useful to extract of microservices. Variability is the ability to create system variants for different market segments or contexts of use [11]. They also observed that microservices extraction can increase software customization. In our system migration, at the ending, we realized that the majority of microservices built were related to a specific business segment of the police intelligence activity.

Agilar et al. [2] has written a systematic mapping study on legacy system modernization. According to the paper, the main contribution of the study is to characterize software modernization according to the existing literature, discussing the related terms, classifying the related research contributions, and presenting the main reasons that motivate an effort of software modernization (also according to the literature). Although the literature presents a huge quantity of publication about software modernization, they generally were proposed without any practical evaluation, in other words, there are just a few studies reporting success experiences in modernizing legacy systems.

The migration process conducted in the Intelligence Department of São Paulo Military Police was not based in a specific method or technique published, but was made in an empiric way according to the real need of modernization. The experience during the migration process helped the team realize that a legacy system can be modernized at first using a monolithic approach and after some time, it will be necessary the use of microservices, that will be performed easily because of a better domain of business learned during the modernization process.

Laigner et al. [24] conducted a study about Big Data legacy System to Microservices Event-Driven Architecture, applying action research, investigating the reasons that motivate the adoption of a microservice-based event-driven architecture, intervening to define the new architecture, and documenting the challenges and lessons learned. Among the conclusions mentioned in the paper, it is worth mentioning that, in their words, defining microservices too early in the development process may yield into a wrong definition. Although microservices have a lot of benefits, if the requirements is not sufficiently mature, it can bring more problems than advantages.

Mazzara et al. [26] performed a case study based on the FX Core system, a mission critical system of Dankse Bank, the largest bank in Denmark. The migration process went through steps similar to those identified in the São Paulo police intelligence department. They also preferred to start the functionalities in a single application unit, which in their words, "allows the team and the organization to uniform the vision, but also the understanding of the specific approach and of the coding standard". After this step, the split into new services will be natural, such as was experimented in this study. Other important observation is the is the "attraction" of microservices architecture with DevOps. Mazzara et al. [26] related

that "devops and microservices appear to be an indivisible pair for organizations aiming at delivering application and services at high velocity". When we started to migrate the monolithic system to the microservices architecture, we were not thinking about DevOps, however, we realized that it would be necessary to implement some tools to optimize the maintenance and deployment process.

Fritzsch et al. [20] investigated the migration process of 14 systems across different domains and sizes. They concluded that the intention of migrate legacy system is to overcome the difficulties of maintainability, which comes with some symptoms such as costs, changes are too costly or are prone to cause side effects and bad analyzability. There are also problems with traceability, long startup times and downtime during updates or difficulties to apply updates in general. This problems is quite similar to what was related about the situation of legacy systems in Intelligence Department. They also analysed why these companies chosen microservices. The main arguments used were scalability of the architecture, development teams, aiming for smaller and better manageable and maintainable units. These arguments also converge with what was reported previously, that is, dividing the system into small application units would facilitate the distribution of responsibility to the development team, in addition to improving the maintainability of each service.

Finally, it is very important to mention the "MonolithFirst" approach of Fowler [19], who alerts that going directly to a microservices architecture is risky. He explains that a monolith allows you to explore both the complexity of a system and its components boundaries. He provides some ways to execute a monolith-first strategy, such as designing a monolith carefully, paying attention to modularity within the software, both at the API boundaries and how the data is stored. It was the way that the monolithic architecture was developed during the legacy system migration and it helped to break our application in microservices, because paying attention in this process made our team have more knowledge about the system functionalities and business areas, consequently, our migration process became easy to perform.

7 Final Remarks

Based on what we have been exposed in this paper, we have figured out that in some time, legacy systems must be evolved, because it becomes very difficult to maintain, understand and is not user friendly. During the migration to modern technologies and architectures, the adoption of microservices architecture is not necessarily the only way of evolution, even though this approach brings a lot of benefits that are mentioned in this study.

The monolithic architecture does not mean that is outdated, in that case, it was a good start point to implement new technologies and service oriented architecture such as REST. It helped us also understand better the business areas of intelligence police department and the others of the Military Police. Thereby, microservices are not adopted as a starting point, but it became an inevitable consequence in the entire migration process, making it became a natural need at

the time when certain difficulties emerged in the monolithic architecture, which began to make the development process costly, whether in the use of human or technological resources.

The initial adoption of the monolithic architecture, taking the necessary care in organizing the source code in the project, allowed for greater ease in the construction of microservices, enabling a clearer understanding of what a microservice actually has to be. It is important to point that all of these observations were confirmed by the development team, independently of experience level.

The final adoption of this architecture allowed the advantages of scalability, flexibility, control, monitoring, build automation, greater involvement of the development team and increased productivity, causing considerable evolution in the systems developed and maintained by the Military Police Intelligence Department of the State of São Paulo.

Acknowledgments. We want to thank the Ministry of Justice and Public Security of Brazil and the Military Police of São Paulo State for supporting this research.

References

1. Abrams, C., Schulte, W.R.: Service-oriented architecture overview and guide to soa research. Technical Report G00154463, Gartner (2008). https://www.gartner.com/doc/575608/serviceoriented-architecture-overview-guide-soa
2. Agilar, E., Almeida, R., Canedo, E.D.: A systematic mapping study on legacy system modernization. In: The 28th International Conference on Software Engineering & Knowledge Engineering, SEKE, USA, pp. 345–350 (2016)
3. Bennett, K.: Legacy systems: coping with success. IEEE Softw. **12**(1), 19–23 (1995)
4. Bisbal, J., et al.: An overview of legacy information system migration. In: Proceedings of Joint 4th International Computer Science Conference and 4th Asia Pacific Software Engineering Conference, pp. 529–530. IEEE (1997). https://doi.org/10.1109/APSEC.1997.640219
5. Blischak, J.D., Davenport, E.R., Wilson, G.: A quick introduction to version control with git and github. PLoS Comput. Biol. **12**(1), e1004668 (2016)
6. Boettiger, C.: An introduction to docker for reproducible research. ACM SIGOPS Oper. Syst. Rev. **49**(1), 71–79 (2015)
7. Brazil: Constitution of the federative republic of brazil (1988). http://www.planalto.gov.br/ccivil_03/constituicao/constituicao.htm
8. Brazil: Decree n 3.695 of December 21 2020. Brazilian public security intelligence subsystem creation (2020). http://www.planalto.gov.br/ccivil_03/decreto/d3695.htm
9. Bucchiarone, A., Dragoni, N., Dustdar, S., Larsen, S.T., Mazzara, M.: From monolithic to microservices: an experience report from the banking domain. IEEE Softw. **35**(3), 50–55 (2018)
10. Carvalho, L., Garcia, A., Assunção, W.K.G., Bonifácio, R., Tizzei, L.P., Colanzi, T.E.: Extraction of configurable and reusable microservices from legacy systems: An exploratory study. In: Proceedings of the 23rd International Systems and Software Product Line Conference, SPLC 2019, vol. A, pp. 26–31. Association for Computing Machinery, New York (2019). https://doi.org/10.1145/3336294.3336319

11. Czarnecki, K.: Variability in software: state of the art and future directions. In: Cortellessa, V., Varró, D. (eds.) FASE 2013. LNCS, vol. 7793, pp. 1–5. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-37057-1_1
12. Davis, A.L.: Spring cloud. In: Spring Quick Reference Guide, pp. 231–246. Apress, Berkeley, CA (2020). https://doi.org/10.1007/978-1-4842-6144-6_18
13. De Lauretis, L.: From monolithic architecture to microservices architecture. In: 2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), pp. 93–96. IEEE (2019) https://doi.org/10.1109/ICSM.1997.624238
14. De Lucia, A., Di Lucca, G., Fasolino, A., Guerra, P., Petruzzelli, S.: Migrating legacy systems towards object-oriented platforms. In: 1997 Proceedings International Conference on Software Maintenance, pp. 122–129. IEEE (1997). https://doi.org/10.1109/ICSM.1997.624238
15. Desmond, B., Richards, J., Allen, R., Lowe-Norris, A.G.: Active Directory: Designing, Deploying, and Running Active Directory. O'Reilly Media Inc., Newton (2008)
16. Dwaraki, A., Seetharaman, S., Natarajan, S., Wolf, T.: Gitflow: flow revision management for software-defined networks. In: Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research, pp. 1–6. ACM (2015) https://doi.org/10.1145/2774993.2775064
17. Erl, T.: Service-Oriented Architecture: Concepts, Technology, and Design. Prentice Hall PTR, Hoboken (2005)
18. Fielding, R.T.: Architectural styles and the design of network-based software architectures. University of California, Irvine, USA (2000)
19. Fowler, M.: Monolithfirst (2015). https://martinfowler.com/bliki/MonolithFirst.html
20. Fritzsch, J., Bogner, J., Wagner, S., Zimmermann, A.: Microservices migration in industry: intentions, strategies, and challenges. In: 2019 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp. 481–490 (2019). https://doi.org/10.1109/ICSME.2019.00081
21. Johnson, R., et al.: The spring framework-reference documentation. Interface **21**, 27 (2004)
22. Josuttis, N.: SOA in Practice: The Art of Distributed System Design. O'Reilly Media, Newton (2007)
23. Knoche, H., Hasselbring, W.: Using microservices for legacy software modernization. IEEE Softw. **35**(3), 44–49 (2018)
24. Laigner, R., et al.: From a monolithic big data system to a microservices event-driven architecture. In: 2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), pp. 213–220. IEEE (2020). https://doi.org/10.1109/SEAA51224.2020.00045
25. Luz, W., Agilar, E., de Oliveira, M.C., de Melo, C.E.R., Pinto, G., Bonifácio, R.: An experience report on the adoption of microservices in three Brazilian government institutions. In: Proceedings of the XXXII Brazilian Symposium on Software Engineering, SBES 2018, pp. 32–41. Association for Computing Machinery, New York (2018). https://doi.org/10.1145/3266237.3266262
26. Mazzara, M., Dragoni, N., Bucchiarone, A., Giaretta, A., Larsen, S.T., Dustdar, S.: Microservices: migration of a mission critical system. IEEE Trans. Serv. Comput. **14**(5), 1464–1477 (2021). https://doi.org/10.1109/TSC.2018.2889087
27. Merson, P.: Microservices beyond the hype: what you gain and what you lose (2015). https://insights.sei.cmu.edu/blog/microservices-beyond-the-hype-what-you-gain-and-what-you-lose

28. Richardson, L., Ruby, S.: RESTful Web Services. O'Reilly Media, Newton (2008)
29. Salleh, S.M., Shukur, Z., Judi, H.M.: Scaffolding model for efficient programming learning based on cognitive load theory. Int. J. Pure Appl. Math **118**(7), 77–83 (2018)
30. Smart, J.: Jenkins: The Definitive Guide. Oreilly and Associate Series, O'Reilly Media, Newton (2011)
31. de Vargas Agilar, E.: A Service Oriented Approach to the Modernization of UnB's Legacy Systems. Master's thesis, University of Brasília, Brasília, Brazil (2016)