ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/326450322

An Experience Report on the Adoption of Microservices in ree Brazilian Government Institutions

Conference Paper · September 2018

DOI: 10.1145/3266237.3266262

CITATIONS
33

READS
1,960

6 authors, including:

Welder Luz
University of Brasília
6 PUBLICATIONS **205** CITATIONS
SEE PROFILE

Marcos Oliveira
University of Brasília
7 PUBLICATIONS **65** CITATIONS
SEE PROFILE

Gustavo Pinto
Federal University of Pará
128 PUBLICATIONS **2,472** CITATIONS
SEE PROFILE

Rodrigo Bonifacio
University of Brasília
77 PUBLICATIONS **842** CITATIONS
SEE PROFILE

All content following this page was uploaded by Gustavo Pinto on 17 July 2018.

The user has requested enhancement of the downloaded file.

An Experience Report on the Adoption of Microservices in Three Brazilian Government Institutions

Welder Luz
Brazilian Federal Court of Accounts
Brasília, Brazil

Everton Agilar
Computing Center
University of Brasília
Brasília, Brazil

Marcos César de Oliveira
Ministry of Planning, Development and Management
Brasília, Brazil

Carlos Eduardo R. de Melo
Ministry of Planning, Development and Management
Brasília, Brazil

Gustavo Pinto
Federal University of Pará
Belém, Brazil

Rodrigo Bonifácio
University of Brasília
Brasília, Brazil

Abstract

Although monolithic applications are still the *modus operandi* of many software systems, the microservices architecture, which favors small and independent applications, is gaining increasing popularity. This is part due to its claimed benefits, which includes better scalability, productivity, and maintainability. However, little is known about how developers and architects perceive the benefits of migrating from monolithic applications to microservices, and what are the challenges towards achieving them. In this paper we discuss the motivation, benefits, and challenges related to the migration *from monolithic enterprise architectures to a microservices based architecture*. We report several lessons learned that arose from a two years process faced by three Brazilian Government Institutions. We also cross-validate these findings with a survey conducted with 13 practitioners in the studied companies. The results of our investigation highlight some evidence that the adoption of microservices brought several benefits for these institutions, such as (a) reducing development time and risks related to deployment activities and (b) increasing the opportunities to experiment with different technologies and development models (such as hackathons). However, our observations reveal that the adoption of microservices is still a challenging task, 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. This study is particularly relevant for institutions interested in adopting a software architecture based on microservices, and we are currently sharing our experiences with other institutions.

CCS Concepts **•Software and its engineering →Software architectures; Software post-development issues; Empirical software validation;**

Keywords monolithic applications, microservice applications, experience report, lessons learned

ACM Reference format:
Welder Luz, Everton Agilar, Marcos César de Oliveira, Carlos Eduardo R. de Melo, Gustavo Pinto, and Rodrigo Bonifácio. 2018. An Experience Report on the Adoption of Microservices in Three Brazilian Government Institutions. In *Proceedings of 32nd Brazilian Symposium on Software Engineering, São Carlos, Brazil, 17 – 21 September, 2018 (SBES'18),* 10 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SBES'18, São Carlos, Brazil
© 2018 ACM. 978-x-xxxx-xxxx-x/YY/MM…$-
DOI: 10.1145/nnnnnnn.nnnnnnn

1 Introduction

Developing monolithic enterprise systems brings several challenges related to maintenance, scalability, and lack of autonomy of development teams to adopt new technologies that diverge from the underlying architecture [6]. Although companies have invested a lot of resources to build monolithic enterprise systems, currently there is a trend to develop systems using the composition of microservices, that is, small pieces of cohesive and autonomous software [20].

Microservices are a relatively new architectural style in the realm of software development strategies. It allow developers to decompose a software in terms of small deployable units, which is in sharp contrast with enterprise system architectures that often comprise a single deployable component. The microservices deployment units are autonomous, and thus they can be deployed in small and isolated services, according to the Single Responsibility Principle [15]. As we shall see in Section 2.1, one of the expected benefits of microservices is the ability to independently deploy individual sets of services, reducing the downtime of other parts of a system. Other expected benefits include improvements on scalability, maintainability, and productivity of development teams [3]. Microservices is also commonly associated as an enabler of other software development practices such as DevOps [4, 12, 14].

In this paper we present an experience report based on a two-years adoption process of microservices-based architecture in three Brazilian Government Institutions, namely:

(a) the Brazilian Federal Court of Accounts (hereafter TCU);
(b) the Brazilian Ministry of Planning, Development, and Management (MP), and;
(c) the Computing Center of University of Brasília (CPD/UNB).

These thee institutions faced different needs that motivated them to adopt microservices, including a rigid monolithic architecture, the lack of freedom from the development teams to explore new technologies, and issues such as code duplication and redeployments of entire applications. It is important to note that these institutions differ significantly from tech companies (we discuss these key differences at Section 2.2) and consequently represent a new dimension of potential microservices-based architecture users that are not yet fully understood. To better understand the challenges that these institutions faced when transitioning to a microservices-based architecture, we performed two studies.

In the first study, we started with a qualitative observational research [9] that aims to "understand how (lead architects) interpret their experiences (with the adoption of microservices)". To this end, we promoted round tables, played development roles, observed and collected information from lead architects of the three institutions, which are migrating from monolithic software architectures to an architecture based on microservices. During the course of this observation, we decided to report their perceptions (and the perceptions of their colleagues) about (a) the motivations that had led decision makers, software architects, and software developers to adopt microservices at their institutions, (b) the technical decisions they have been taking, and (c) the challenges they have faced during the migration process.

In the second study, we conducted a survey with software architects and developers of the same three institutions, in order to cross-validate the results of the first study. The goal of this survey was to (1) proper evaluate, after the core of the adoption process, the perceived benefits and challenges faced and (2) to quantify whether or not our observations from the first study were accurate. To this end, we elaborated an online questionnaire that we made available during two weeks to our colleagues of the three institutions that also participated (less actively, though) in the migration process.

Altogether, this paper makes the following contributions:

- A two years observation from inside three different institutions that were dealing with the need to migrate towards microservice-based architecture;
- A survey with 13 practitioners that promoted and participated in the migration towards microservice-based architecture;
- A discussion of the obtained results in terms of lessons learned and claims potentially relevant for further investigations.

2 Background

Here we provide the grounds for understanding microservices-based architecture (Section 2.1), and some descriptions about the institutions that we conducted our study as well as the target software systems that motivated the migration towards microservices-based architecture (Section 2.2).

2.1 Microservices-based architecture

Microservices are a relatively new architectural style in the realm of software development strategies. It allows to build a software application design as suites of services that are independently deployable. In contrast, Enterprise Applications are typically built as a monolithic deployment unit, that is deployed in a application server and every change in the code demands a new build and deploy of the entire application. Figure 1 presents an example of a video sharing platform using a monolith deployment unit (on the left), and the in the form of microservices (in the right).

In the microservices-based architecture, each unit (or service) has a specific role, and a service communicates to another service often over the HTTP protocol. This architectural style could yield several potential benefits, including:

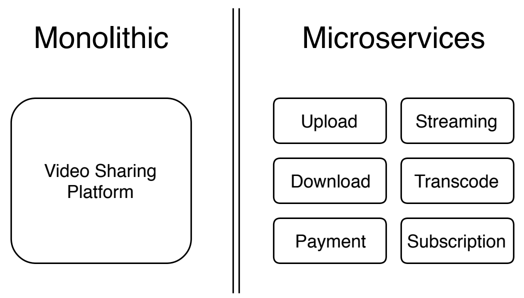

Figure 1. An example of a monolith application (on the left) and a microservices application (on the right).

Independent development. Since the development is now concentrated in small, independent units of working software, software teams can work them independently. For instance, one software team can work on the "Payment" unit without interfering or even knowing the existence of the "Download" unit. This separation of services might incur in less conflicts and communication issues. Moreover, a given unit can be implemented in a more appropriated programming language, with enables developers to fully explore the benefits of other programming languages, frameworks, and tools.

Independent deployment. Since the units are small and independent, and potentially written in different programming languages, they can be deployed in different containers and services at different times. This benefit fosters rapid releases of features, configuration changes, or bug fixes in a sustainable and efficient way [12]. As an example, there is no need to suspend the whole system activity to deploy the fix of a bug at the "Subscription" unit.

Independent scalability. Since deployments are independent, each unit can be scaled independently. If one unit is facing more demands than another unit (e.g., if a new movie is just released, the "Download" unit might be more demanded), such a unit can be scaled up, e.g., in an on-demand cloud platform, to handle the increased load. There is also a monetary gain in this approach: since one does not need to scale up the entire application, a fewer number of virtual machine (or containers) instances might be needed to meet demand.

Although many other benefits have been claimed and discussed (particularly in the grey literature[1]), such as code reusability and maintainability, decreases the learning curve, avoids long-term commitment to a single technology, there are few studies assessing whether these benefits are indeed perceived in practice and, if so, what are the challenges that software companies face to achieve them. The goal of this work is to provide more empirical evidence that could support (or even refute) the use of microservices-based architecture.

2.2 Studied Institutions

The three institutions where we developed our work are public, well-known, and long-lived Brazilian institutions. As any public institution in Brazil, most of their budget come from the Federal

[1]For instance, https://dzone.com/articles/benefits-amp-examples-of-microservices-architectur

and/or State budget. As a consequence of the current scenario of limited budget, aligned with the conservative political views that drive these institutions, such public institutions often do not have the freedom to explore new software solutions, making them hostage of aged, ill-tailored technology stack. Moreover, since the employees of these institutions have permanent stable positions, these institutions they hardly face the issues found in Silicon Valley tech companies (e.g., that face issues related to job rotation [17]) or open source communities (e.g., that face issues related to bus factor [7]). These facts make the studied institutions particularly relevant to the scope of this study since the lessons learned from prior work might not be easily transferred to them.

As another reason towards the use of microservices, this architecture was perceived as beneficial since the development team do not need to throw away the existing system in order to replace for a newer software solution (which would be a luxury prohibitively expensive). Instead, the software team could steadily introduce new services that could replace small parts of their software systems, e.g., following the strangler application pattern [18]. In this pattern, a monolithic architecture is slowly replaced with a more componentized one. During the transition process, the new componentized architecture may delegate tasks to the system it is replacing. Over time, the old system is strangled in favor of the new one.

Although all studied systems are proprietary by nature, we could provide some quantitative details about the SIOP system, which is develop by MP (more details at Section 4.1.2). This monolithic system is one of the most important financial systems of the Brazilian Government, which two of the authors have contributed significantly. It is written in Java, and is under development since 2009. It has 521KLOC, 241 packages, and 5.6K classes. Regarding code contributions, it has more than 12K commits and 3K issues.

3 Method

In this section, we describe the research questions (Section 3.1) and the research method we used in the two conducted studies (Section 3.2 and Section 3.3).

3.1 Research Questions

As a first step towards our research goal, we designed the following research questions:

RQ1. What are the reasons that motivate each institution to adopt a microservice-based architecture?

This first research question explores the different reasons that motivated the studied institutions to explore other possible architecture alternatives, including the microservice-based one. This research question is particularly relevant due to the context within our studied institutions are settled, which is far from representing a traditional software development company.

RQ2. Which were the main technical decisions taken to leverage the microservice-based architecture?

In this research question our goal is to uncover the different technical decisions that the software development teams had to made in order to leverage a microservice-based architecture. Since such a transition had to be made smoothly and with minimum overhead, one might expect that the technical decisions were made in a conservative way (e.g., employing a similar software stack). In this research question we explore this belief.

RQ3. What are the benefits the institutions perceived during the adoption of microservices?

RQ4. What are the challenges the institutions faced during the adoption of microservices?

Finally, these two set of questions (RQ3–RQ4) are intended to uncover some benefits related to this adoption and hidden challenges that were perceived throughout this process.

To answer these research questions, we conducted two studies: a self-observational study and a survey with other developers that participated in the transition.

3.2 Study 1: A Self-Observation

In this first study, we report an observational study. In this method, the research is both an observer and a participant in some activities. Although software engineering researchers are only recently taking advantage of it [13], this method is well-known and well-used in other disciplines [9]. In spite of the fact that this method has many strengths (e.g., it can provide unique set of views that could not be perceived from external eyes), it also has limitations (e.g., it presents a single perspective and it is rather subjective). To mitigate some of these limitations, as we shall see in Section 3.3, we triangulate the findings of this study with a survey answered by other participants that also played important roles in these institutions.

In this observational, self-examination study, we describe a **24 months participation** of ourselves playing the role of software architects in the three studied institutions. To study our experiences, we had several meetings and round tables with other institutions' representatives, archived dozens of emails and several diaries, and performed many informal interviews with colleagues. We then analyzed the results of the meetings, round tables, emails, and interviews to confirm or refute challenges and claims regarding the transition to a microservice-based architecture.

Our diaries and emails spanned the work from 2016 to 2018. The three institutions joined the transition to microservices at different time-window and were motivated by different needs (Section 4.1 discusses the needs in details). However, the three institution referred to the University of Brasília (UnB) in order to further substantiate their background and, consequently, their technical decisions that could lead them to a better architecture. A researcher from UnB was then dedicated to support the institutions to achieve their goals. The observations reported in this paper represents the point of view of one researcher (that also played a software architect role) and three software architects (that became more involved with software engineering research).

3.3 Study 2: Survey with Developers

After we concluded a relevant part of the transition towards the microservices-based architecture, we conducted an online survey with developers of the three institutions. The goal of this survey was to enrich our understanding about the consequences of the architectural migrations discussed in the following sections of this paper. In particular, we aimed to understand the perceptions of software architects and developers of the three institutions about the adoption of microservices in their environment.

The target population of this second study are software developers that have contributed to the migration effort or that have developed microservices there. Based on our initial knowledge of the adoption of microservices in the three institutions (as one of the outcomes of the first study), we asked the following 10 questions:

(Q1) Do you agree that the adoption of microservices contributes to technological innovation?
(Q2) Do you agree that the adoption of microservices makes the deployment activities more flexible?
(Q3) Do you agree that the adoption of microservices leverages continuous delivery?
(Q4) Do you agree that the adoption of microservices increases software availability?
(Q5) Do you agree that the adoption of microservices is necessary to successfully introduce DevOps techniques in you institution?
(Q6) Do you agree that the adoption of microservices reduces the *time-to-market* to introduce new software features?
(Q7) Do you agree that the (lack of) criteria to decompose a system into microservices still challenges the adoption of this architectural style?
(Q8) Do you agree that it is still hard to convince decision makers about the benefits of adopting microservices?
(Q9) Could you enumerate other benefits related to the adoption of microservices?
(Q10) Could you enumerate other challenges related to the adoption of microservices?

Questions (Q1)–(Q8) are closed questions whose available options are based on a Likert scale (*Strongly Disagree* (1), *Disagree* (2), *Neither Disagree nor Agree* (3), *Agree* (4), *Strongly Agree* (5)). Questions (Q9) and (Q10) are open-ended questions. We qualitatively analyzed the answers to the open-ended questions.

We designed an on-line survey (it can be found at: https://goo.gl/dUQT6k), and asked developers from the three institutions to answer it. Before sending the survey, we validated it internally in order to catch wording problems, or improve the title of the questions to ease understanding. We sent the survey to 33 employees of the three studied companies that have participated in the migration process. During a period over two weeks, we obtained 13 answers. Among the respondents, 12 of them software developers, though one of the respondents also works as an infrastructure manager.

We analyze the answers of the closed questions using plots, tables, and distributions. For the open questions, we opted for a straightforward qualitative approach: we grouped each answer into categories; after the initial pass, we refined the categories in order to find broad and more meaningful groups. We use these answers to cross-validate the findings of our first study.

4 Results for Study 1: Observations

In this section we discuss the scenario in the Brazilian institutions that motivated them to migrate to a microservices-based architecture (Section 4.1), we elucidate the adoption strategies that the institutions employed to leverage the microservices-based architecture (Section 4.2), and we also present some initial progress that the institutions made in order to migrate representative software systems to microservices-based architecture (Section 4.3).

4.1 Before the adoption

This section presents the different reasons that led the three institutions (TCU, MP, and CPD/UnB) to start a process to adopt microservices as an architectural style.

4.1.1 TCU Obsession for Technical Conformance

For several years, *technical conformance* was one of the main architectural requirements at TCU. In this scenario, all enterprise systems should be developed considering a shared database, a shared domain model, and a common stack of technologies, programming languages, libraries, and tools (also using specific versions). In summary, the mentioned stack was based on the *Java Enterprise Edition* specification, using *Java Server Faces* for the presentation tier, *Enterprise Java Beans* for the domain tier, and *ORACLE* as the underlying relational database system. However, in recent years, it became clear that this stack was not an interesting option to solve several classes of problems at TCU, which could be better addressed reducing the coupling with the shared database and domain model components and using different development platforms, different programming languages, or even different persistence layers. However, the requirements of technical conformance often led to an unusual design for specific situations, increasing the development costs and time to market. There was also an issue related to the lack of development team motivation, since the development teams were not free to experiment with new technologies. Therefore, **reducing the coupling with a shared domain model component and making technical innovation feasible** were the principal reasons for experimenting a microservices-based architecture at TCU, together with the well-known possible benefits, such as more independent deployment processes.

4.1.2 MP Nightmares with Builds and Deployments

The MP is responsible for the main design decisions and development of the *Brazilian Integrated Budget Planing System* (SIOP), a monolithic enterprise Java system that supports all workflows related to the Brazilian Federal Annual Budget. In a first moment, the design of SIOP was based on a typical *Java Enterprise Edition* (JEE) application (similar to the reference architecture at TCU).

Also similar to TCU, the monolithic and rigid architecture hindered the adoption of new technologies, programming languages, and tools. The development team claimed that even simple tasks (such as upgrading the application server or the version of the frontend components) were real nightmares, in particular because any changes would have the potential of affecting the whole system. Moreover, the deployment process of SIOP was also a challenge itself, because it required to stop the application server to update the version of a single *Enterprise Archive* component that comprises the entire code base of SIOP (~521KLOC).

To reduce risks and downtime of the application, the infrastructure team was really conservative with respect to the specific moments in which a deployment might happen. Still, SIOP has a heavyweight development process, which is a problem since several business areas present a high demand for frequent deployments—particularly when approaching deadlines of the federal budget planning and execution cycle. Therefore, the main motivation for implementing a microservices-based architecture at SIOP was to **allow the adoption of new technologies** in independent parts of the system and, consequently, to **make the deployment process more flexible and independent**, in such a way that it would become possible to deploy a business subset of SIOP independently of the remaining parts of the system.

4.1.3 CPD/UnB and its Duplicated Systems

The enterprise ecosystem at CPD/UnB involves the implementation of different academic systems using (a) different architectures (from two-tier systems to Web systems) and (b) programming languages (such as Java, Visual Basic, and C#). That is, architectural conformance was not a significant issue for CPD/UnB. However, the lack of a rigorous approach for software development led to problems in different situations. For instance, a student should be able to register himself into a course using either a desktop or a Web based application of the same system. Nevertheless, the business logic related to the registration process was implemented in both versions of the system. The maintenance costs were excessive because bug fixes and implementation of new features have been often modifying not only different parts of a system, but also different systems. In this chaotic scenario, the adoption of microservices appeared as a possible solution to **decrease code duplication**.

The microservice adoption was perceived as a reasonable goal, since instead of having the entire legacy system replaced with a new major release, the development team could modernize the codebase through the creation of small services (e.g., performing the modernization of the systems in small increments). This would allow developers to migrate parts of a given system A to a set of microservices, and then make the other systems that depend on A to start to consume those services.

Summary of RQ1: The reasons that motivate the institutions to adopt a microservice-based architecture varied from (1) reducing the coupling with a shared domain model component and making technical innovation feasible, (2) decreasing code duplication and maintenance efforts, and (3) allowing the adoption of new technologies in independent parts of the system and to achieve independent deployment.

4.2 Adoption Strategies and Technical Decisions

As aforementioned, the goal at TCU was to increase both innovation opportunities and the autonomy of development teams to choose suitable programming languages and tools for solving particular problems. Without any strategic plan, the development team started a new project using a microservices-based architecture. Though, to avoid drastic changes and reduce risks, that team decided to maintain a Java based infrastructure. The first benefit found with the adoption of microservices was the reduced coupling between services and the legacy backend. Afterwards, another development team decided to decompose a monolithic system into microservices and thus the microservices adoption disseminated to different teams.

Together with the increasing knowledge on microservices development, the teams enlarged the set of tools to develop microservices, such that it became possible to select the best alternative for a given situation. The current perception is that the adoption of microservices reduced the risks related to technical innovation. In this context, different databases, programming languages, programming libraries, and tools have been explored as backend alternatives. Similarly, the development teams already explored alternatives for the frontend. Versions of libraries or even the development platform have been managed with high flexibility, which represents a great contrast to the previous model. The development teams also increased their autonomy with respect to deployment activities, which started to occur more frequently and with an increasing degree of automation. Well-defined services boundaries enabled this benefit, and thus the development team perceived that the deployment of a microservice often occurs in an isolated manner.

The current architecture at TCU uses a continuously updated *service central repository* that is fed by specific jobs that collect services metadata from each microservice. Regarding deployment, each microservice runs in the context of a specific (Docker) container. Each commit in the source code repository enables a continuous deployment pipeline that builds a new container image with the new version of a microservice, and pushes this image to an internal registry. Regarding security, clients of a microservice must be authenticated using tokens that are sent in the request headers. Each microservice has its own credentials (user name and password), which allows it to obtain an authentication token before consuming another service.

Due to the problems discussed in the previous section, the development teams at MP established a working group to propose a new architecture based on microservices. The first decision was to investigate the adoption of well-known services and microservices enabling technologies (such as Apache Thrift and Google Protocol Buffers) complemented with proprietary tools. Another relevant decision was to keep Java as the main backend programming language, without using Enterprise Java Beans (EJBs), however. Instead, the development team proposed Duna (https://github.com/duna-project/duna-poc), a framework that provides EJB features similar to those provided by the EJB specification (e.g., transaction management and service communication), but works independently from the application server. Regarding the frontend technologies, Javascript MVC-clients are being used. The communication between the frontend and the backend is primarily carried out using either GraphQL or REST-based calls.

After these first definitions, the development team implemented a pilot project to understand the benefits and consequences of adopting the proposed architecture. The pilot project involved the migration of a non-trivial "module" of SIOP, responsible for a specific process of the system, which requires an integration between the Executive and Legislative Brazilian powers. The promising results of this pilot project encouraged the development team to proceed with the migration process of other modules of SIOP.

The current architecture at MP uses Docker and Docker compose for services' provisioning and discovering. The integration between microservices is supported through requests that come from the user interface or using immediate messages between the backend components, which speedups the execution of some transactions. Similarly to the TCU architecture, the authentication is based on tokens, though using only user credentials. The architecture also uses the Apache HTTP Server for dealing with loading balance among Docker containers.

Finally, our migration case at CPD/UnB was supported by a set of research and development initiatives, which started when a software architect from CPD/UnB registered himself in a graduate course at UnB, aiming at designing a software modernization process to the institution. After a systematic mapping study on software modernization [1] and a better understanding of the institution needs, the development teams decided to use microservices on top of an "in a house" software infrastructure. Such a decision was motivated by the lack of resources to invest either in software licenses or training to adopt an "*off the shelf*" solution and by the perception that developing the initial set of tools and libraries would be feasible, would favor comprehension of the whole stack, and would allow the teams to tailor the technologies to the particular needs of the institution.

In this scenario, the development team created ErlangMS (https://github.com/erlangMS), a microservices-based infrastructure developed in Erlang that supports the implementation of services using different programming languages. ErlangMS addresses the *polyglot requirement* of building software using different languages at the institution. To evaluate the architecture, the development team developed a relatively small pilot project (almost 5KLOC). The goal of this pilot project was to modernize a legacy *social student assistance system*. The success of this pilot project was also important to convince high level decision makers to migrate part of the existing technological infrastructure.

Although the managers and developers involved in this pilot project considered it a success, the pilot project was not deployed until the development team gave evidences that the architecture was able to solve a scalability issue of the *single sign-on* authentication mechanism used in the entire ecosystem of the institution. Again, in this situation, another proof-of-concept based on microservices, on top of ErlangMS, was designed and the development team demonstrated to the decision makers that the solution could mitigate the scalability problems of the legacy infrastructure.

After that, the architects and development teams at CPD started to adopt ErlangMS in the production environment (still side by side with the legacy systems). The current decision at CPD/UnB is to run multiple instances of ErlangMS, each one working as a container for a small number of microservices. Using the distribution facilities of the *Erlang Open Telecom Platform* (OTP), the different instances of ErlangMS communicate to each other using Erlang messages, which simplifies the implementation of a number of concerns, such as services discovery and orchestration.

Summary of RQ2: The main technical decisions are related to (1) implementing a pilot project to understand the adoption strategy o micro-serivces in a given institution, (2) reusing components of the legacy systems to implement microservices and reduce risks, and (3) taking advantage of containers to enable the deployment of microservices in different environments (e.g., test, homologation, and production).

4.3 After the adoption

The TCU development team has already migrated two legacy systems to the microservices based architecture (they comprehend 6 microservices and 40KLOC). In addition, four new systems have been implemented using microservices. Altogether, there are more than 30 microservices running on the production environment at TCU. However, the current decision is to not migrate any other legacy system to the microservices based architecture, since this process was considered time consuming and costly. The current understanding is that only new systems should be designed using the microservices-based architecture.

Differently, only one module of SIOP has been migrated to the microservices-based architecture, comprising 12 microservices and almost 20KLOC. The current understanding of the MP development team is that the entire system should be migrated to the new architecture. That is, although being a challenging task, the perceived benefits justify the efforts. New modules should be migrated in the following months. However, it is clear that, to migrate the whole system, it would be necessary a long term effort (in terms of years)—though the high level decision makers believe that this is a strategic decision.

Item	low	neutral	high
Deployment	7.69	0.00	92.31
ContinuousDelivery	7.69	0.00	92.31
Innovation	0.00	15.38	84.62
TimeToMarket	15.38	7.69	76.92
Availability	15.38	15.38	69.23
DevOpsEnabler	0.00	30.77	69.23

Table 1. Answers' distribution for questions (Q1)–(Q6)

Finally, the development team at CPD/UnB has developed several microservices for dealing with the authentication, authorization, and monitoring mechanisms of the corporate systems using microservices (comprehending 4 services and almost 2KLOC). In addition, more than 20 microservices (almost 15KLOC) that either support business workflows or share university data have also been implemented. This shift promoted new development models (such as crowdsourcing or hackathons) that were not envisioned before. More recently, a whole legacy system was migrated to the microservices-based architecture, and the results are promising. As a result, the high-level decision makers started to invest in the new platform.

5 Results for Study 2: Survey

We present the results of our survey grouping questions (Q1) – (Q6) and (Q9), which focus on the benefits achieved using microservices and questions (Q7), (Q8), and (Q10), which focus on the challenges developers faced with the adoption of an architecture based on microservices.

5.1 Benefits Related to the Adoption of Microservices

As Figure 2 and Table 1 show, almost all respondents agree that the adoption of microservices brings benefits to continuous delivery, make deployment more flexible, leverage innovation, reduce time-to-market, enable DevOps adoption, and increase software availability.

In more details, respondents consider that microservices contribute most for leveraging continuous delivery and making deployment activities more flexible. Regarding these two characteristics, 53% of the respondents strongly argue in favor of microservices and 38% argue in favor of microservices. We also found that 30% of the respondents either *disagree* or *neither agree nor disagree* that microservices improve software reliability (against 70% that either *agree* or *strongly agree* that microservices improve software reliability). Therefore, increasing software reliability is not the most perceived benefit that result from the adoption of microservices. In addition, 30% neither *agree nor disagree* that the adoption of microservices is a necessary step for the DevOps adoption (against 53% that *agree* and 15% that *strongly agree*). A possible reason for

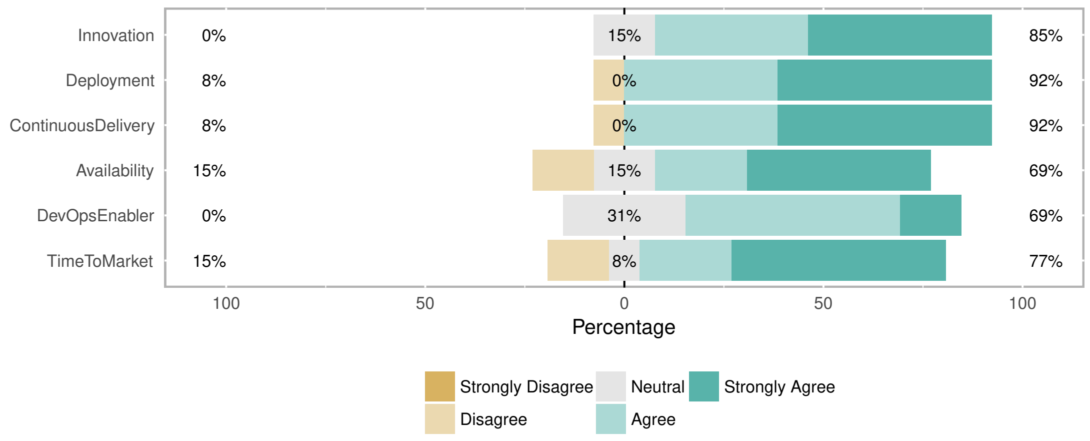

Figure 2. Plot with the answers' distribution for questions (Q1)–(Q6). 1 means Strongly Disagree, 2 means Disagree, 3 means Neutral, 4 means Agree, and 5 means Strongly Agree.

this result is that both techniques (DevOps and microservices) actually complement each other. That is, it is hard to introduces a microservices-based architecture without increasing the automation of deployment tasks and reducing the boundaries between the development and production teams (two assumptions related to the DevOps approach). In this way, DevOps might be actually an enabler approach for the adoption of microservices (instead of the reverse), and for this reason, microservices make clear the needs for adopting DevOps.

Also interesting, 22% either *disagree* or *neither agree nor disagree* that the adoption of microservices reduces the time-to-market of new software features (against 23% that *agree* and 53% that *strongly agree*). Based on the results of the previous study, this perspective might change according to the institution. We have practical evidences that the adoption of microservices increases the production of software at CPD/UnB. The same result might not be true for the other institutions. Finally, 15% *neither agree nor disagree* that the adoption of microservices improves the opportunities for technical innovation (against 38% that *agree* and 46% that *strongly agree*). Altogether, we found some evidences that the adoption of microservices brings benefit to all the mentioned aspects in our survey.

Six developers answered (Q9), the optional open-ended question asking for further perceived benefits of adopting microservices. The respondents included additional benefits such as *team motivation*, *increasing ability for working in small and parallel tasks*, *technology independence*, *support for third party development* and *horizontal scalability*.

Summary of RQ3: The benefits related to microservices usage include (1) the adoption of continuous delivery, (2) the reduced time to market, (3) the increased software availability, (4) the improvement on teams' motivation.

5.2 Challenges Related to the Adoption of Microservices

According to the respondents, the main challenge related to the adoption of microservices is the lack of understanding on how to decompose an existing monolithic enterprise system into a number of microservices (Q7). Differently, based on the answers to our survey, 30% of the respondents did not agree that the lack of involvement of decision makers is a challenge to the adoption of microservices (Q8). We summarize these results in Figure 3 and in Table 2.

Item	Low	Neutral	High
Lack of a Decomposition Criteria	0.00	0.00	100.00
Decision Makers Involvement	7.69	23.08	69.23

Table 2. Answers' distribution for questions (Q7) and (Q8)

The respondents included additional challenges as answers to question (Q10). For instance, one of the respondents pointed out that "*the strong coupling among the components of the legacy system*" hinders the migration process from monolithic to an architecture based on microservices. Other respondent stated that the migration process has introduced new issues related to security "*. . . and radical changes on the development process*". Another challenge relates to the "*increasing degree of automation*" that is necessary to "*put everything to work*", according to one of the respondents. Besides strengthen the challenges related to the lack of criteria (a) to decompose a problem into microservices and (b) to evaluate quality properties of a microservice (such as size and cohesion), one of the respondents also emphasized that"*the use of microservices lead to an unstable distributed environment that brings new challenges related to distributed transactions*".

Summary of RQ4: Among the challenges, we observed that the lack of understanding on how to decompose a monolithic system into a service plays a role. Moreover, some respondents reported the strong coupling among the components of the legacy system and the radical changes in the development process as some additional challenges.

6 Discussion

In this section we summarize the lessons learned from both studies and present our perception with the adoption of a microservices-based architecture in the three institutions.

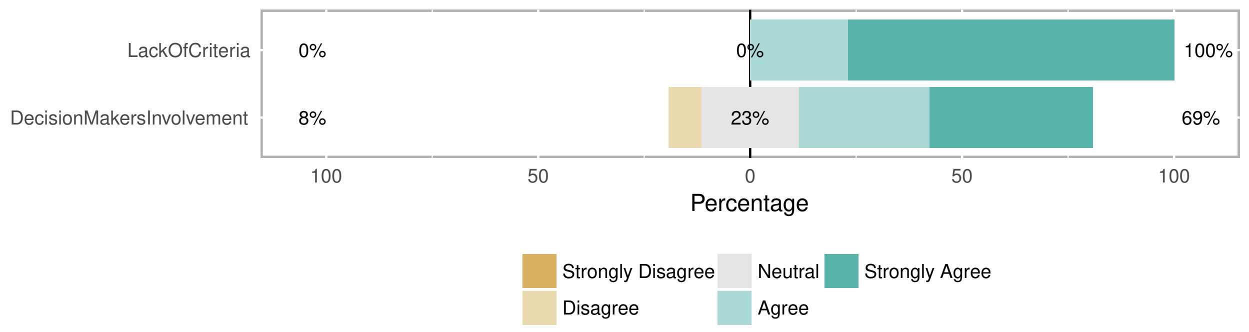

Figure 3. Plot with the answers' distribution for questions (Q7)–(Q8). 1 means Strongly Disagree, 2 means Disagree, 3 means Neutral, 4 means Agree, and 5 means Strongly Agree.

6.1 Lessons Learned

The adoption of microservices at TCU brought many benefits, including a lower coupling with the shared domain model and the increased autonomy of the development teams with respect to technical innovation [5]— a benefit that has been previously discussed [4]. Currently, when one needs to build a new system, the development team is free to discuss and select the set of tools and technologies that better fits their requirements. The microservices architecture also promoted the adoption of DevOps practices, such as Continuous Delivery, which in turn reduced time-to-market and improved deployment activities. Although the deploy of monolithic systems still complies with rigid time frames (once a week), the deployment of microservices might occur several times a day. Besides those benefits, the current decision is to only use the microservices-based architecture to develop new systems. That is, no additional migration effort should take place in a near future.

Continuous delivery in the production environment was considered the main benefit of introducing microservices at MP. Using microservices, it is possible to implement and deploy changes (such as bug fixes or the development of new features) and then put such a change in a production environment with a flexible schedule and a reduced downtime of the entire system. With the adoption of microservices, developers were able to deploy a module in less than a minute (about 10 times faster, when compared to the monolithic system). This was not feasible before. In the current scenario, it is possible to deploy several pre-release versions of a microservice-based module in a single day. Since microservices promote independent software systems, the development teams are also able to experiment with new technologies. For instance, it is possible to change the user interface technology of a given service, without having to propagate this kind of modification to other parts of the system. This benefit allowed the development team to explore suitable technologies to the different areas of SIOP.

Finally, at CPD/UnB, the main benefit with the adoption of a microservices-based architecture is that currently, the development teams at CPD/UnB were able to implement and deliver small pieces of functionalities in short periods of time and implement modernization initiatives using an incremental approach. Also, they were able to migrate parts of a system to microservices that coexist with the other parts of a system that have not been migrated. Microservices also promoted new development models (such as hackathons), fostering external developers (e.g., students) to collaborate through the design and development of new solutions to the institution. Although the modernization of the legacy systems has not occurred as initially planned, it is important to note that the decision to postpone the modernization efforts was not related to the proposed architecture, but instead due to the increasing software demand that delays the execution of the software modernization efforts. We summarize some relevant observations from our study in Table 3.

6.2 Our perception

Besides the benefits discussed above, there are several uncertainties related to migration efforts. For instance, it is hard to convince decision makers from our studied institutions to invest on efforts like this. Based on our experience, it is easier to introduce microservices when developing new systems. The challenges to migrate to a microservices based architecture also increase because microservices and DevOps should be introduced side-by-side, to allow development teams to manage an increasing number of services. As a consequence, the number of tools involved increases substantially, raising the expectations for more skilled teams.

Finally, it is difficult to accurately identify the boundaries of a microservice, and thus we advocate an agile approach for designing microservices. There is a trade-off related to microservices granularity. If a microservice is too small, there will be a lot of deployable unities to manage. However, if a microservice delivers many functionalities, the expected benefits might be compromised. To better understand (and eventually mitigate) these problems, we plan to explore some design recommendations (such a bounded context [11]), and tailor them to the microservices architecture. In the cases we did not find an interesting design for a microservice, we refactor the design using transformations (such as split class and move methods) tailored to the microservices architectural style.

6.3 Limitations

First, since we were observing our own behavior (part of the authors of this paper are the lead architects that promoted the migration towards microservice-based architecture), we may have missed or oversimplified some of our perceptions. To mitigate this threat, we conducted an additional survey with other practitioners in the same institutions that also participated, but did not employ a key role, in the transition towards microservice-based architecture. This survey helped us to understand and cross-validate some of the initial findings. Second, as we aforementioned, our survey was deployed to

Table 3. Summary of the study

Institution	Motivation	Adoption strategy	Benefits	Challenges
TCU	To make technical innovation feasible	Ad hoc; Used well-known tooling	Reduced risks related to technical innovation; Continuous delivery;	Time consuming; Microservices only for new systems
MP	To allow the adoption of new technologies	Pilot project; Designed their own tooling	Continuous delivery; Reduced down time; Explored new technologies	Long term effort
CPD/UnB	To decrease code duplication; Low budget to invest on software licenses	Pilot project in cooperation with an University; Designed their own tooling	Reduced time-to-market; crowdsourcing and hackathons	Hard to convince decision makers

our colleagues in the studied institutions. Although we made them comfortable to decide whether or not to answer the survey, some of them might feel obligated to do so. As mitigation, our survey was designed to favor anonymity in a way that we could not trace back the answer and map to the respondent. Moreover, we decided not to share the survey with other practitioners in other institutions in order to limit our results to the studied institution and the context that motivate them to migrate to microservice-based architecture. As consequence, this study reports the perceptions obtained in three governmental institutions in Brazil. It does not cover other institutions either in Brazil or abroad, neither the perceptions from software companies. It is important to highlight that governmental institutions in Brazil are intrinsically different than, for instance, software companies or tech startups, in particular due to the lack of freedom that rules the institution, which reflects in the conservative technical decision adopted by them. Finally, although we did not focus in any software stack in order to leverage microservice-based architecture, our technical decisions were grounded by the team expertise, and, therefore, might not generalize to other software teams facing similar issues. We welcome replications of our study.

7 Related Work

We found three recent literature reviews on microservices architecture. The first work of Alshuqayran et al. [3] reports the common benefits and challenges of this architectural style, based on the analysis of 33 primary studies. The second of Di Francesco and colleagues [8] presents the analysis of 71 papers reporting research trends, research focus, and the potential for industrial adoption of existing research results on microservices. Finally, the work of Vural et al. [21] reviews 37 research works to investigate the type of research, the motivations behind the current research on microservices, and the emerging standards on microservices solutions.

According to Alshuqayran et al., the commonly agreed benefits on this style include (a) an increasing adoption of agility practices and developer productivity; (b) an increasing software resilience, scalability, reliability, maintainability, and separation of concerns; and (c) a reduced efforts and complexity for software deployment [3]. According to the authors, the main challenges are related to communication / integration, service discovery, performance, fault tolerance, security, tracing and logging, application performance monitoring, and deployment operations. The study calls attention to the emerging nature of the theme due to the lack of experience reports and opinion papers in literature.

In the work of Di Francesco and colleagues [8], the authors report that “Solution Proposal” is the prevalent research category on microservices. According to the authors, this might indicates that the microservice architectural style is still in its infancy. The study indicates that the target problems—complexity, low flexibility, resources management, and service composition- are consequences of the existing trade-offs with the microservices adoption. Although microservices bring the benefits of high flexibility, the inherent complexity to manage a high number of distributed services is still a challenge. The study also suggest that there is a gap between the research on microservices and the needs of the industry, mainly because the current research efforts mostly focus on proposing solutions to particular needs. Similarly to the previous mentioned work, Vural et al. state that “Solution Proposal” is the most widely research category explored on microservices [21], followed by validation research and evaluation research. The paper also presents REST as an important emerging standard and Docker as the most frequent used tool on microservices studies.

There is a number of papers addressing the question of decomposing monolithic applications into microservices, which indicates that this is an issue related to microservices architecture adoption. The decomposition strategies vary. For instance, Ahmadvand et al. present one approach to identify microservices candidates based on requirements analysis [2], whereas Furda et al. present a strategy that, starting from explanations about three specific challenges in decomposition task (multitenance, statefulness and data consistency), point out solutions to them using well known SOA patterns. Other studies explore code clusterization as a possible strategy to identify microservices candidates from a monolithic legacy system [10, 16]. In fact, decomposing existing monolithic applications into microservices is a challenge (as we reported in this paper). Nevertheless, our experience reveals that decomposing a new system using microservices is also a challenge, and we could not find substantial related work on this.

Other research works deal specifically with the relation between microservices and DevOps. For instance, Balalaie et al. [4] presents an experience report about the adoption of microservices architecture. According to the authors, the adoption of microservices enables the use of DevOps related practices, including continuous integration and deployment pipelines, continuous monitoring and vertical division of project members into cross-functional teams. Another close related study is presented by Taibi and colleagues [19]. The authors conducted a survey, interviewing 21 practitioners, who adopted a microservices based architectural style, to elicit the current motivation and specific issues in their adoption. According to the survey, maintenance was always reported and rated as very important by all the participants. Scalability, delegation of responsibilities, and the easy support for DevOps also frequently drive adoption, while the main issues are related to the process to (a) decompose a monolithic system, (b) migrate and split data in legacy databases, and (c) integrate different services.

8 Conclusions

In this paper we report a two-years experience introducing microservices in three Brazilian Government Institutions (TCU, MP, and CPD/UnB). Microservices is an emerging technology that advocates the decomposition of a software into a "*suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API*" [20]. Independent deployment and reduced downtime are two expected benefits with the adoption of microservices. However, these benefits have been previously discussed in the context of *high-tech companies*, and here we go beyond this recurrent perspective, and present that the adoption of a microservices-based architecture, even in rigid, government institutions, brings additional benefits with respect to

Innovation: the adoption of microservices allowed the introduction of (a) programming languages and tools that best fit the needs of particular problems and (b) new development models (such as crowdsourcing and hackathons). We observed this benefit in the three institutions.

Time-to-market: the adoption of microservices reduced the time-to-market to introduce new features into existing systems and to develop entire applications. We observed this benefit at TCU and CPD/UnB.

Motivation of development teams: Before adopting microservices, the enterprise systems of the three institutions have been developed using out of date technologies, which compromise the development team motivation. The adoption of microservices attenuate this problem in the three institutions.

Nevertheless, it is difficult to break a monolithic system into microservices, mostly because there is a lack of guidelines with more precise descriptions about the expected granularity of a microservice. In addition, breaking a monolithic system into microservices demands the introduction of new practices and tools for managing an increasing number of deployable components. This might lead to a technological disruptor. Finally, MP and CPD/UnB have decided to continue the modernization effort. Differently, after the migration of two legacy systems, TCU have decided to only use microservices in the development of new systems.

Acknowledgments

We thank the 13 participants in our survey and the reviewers for their insightful comments. This work is partially supported by CNPq (406308/2016-0).

References

[1] Everton Agilar, Rodrigo Bonifácio, and Edna Canedo. 2016. A Systematic Mapping Study on Legacy System Modernization. In *SEKE*. KSI Research Inc. and Knowledge Systems Institute Graduate School, 345–350.

[2] Mohsen Ahmadvand and Amjad Ibrahim. 2016. Requirements reconciliation for scalable and secure microservice (de) composition. In *Requirements Engineering Conference Workshops (REW), IEEE International*. IEEE, 68–73.

[3] Nuha Alshuqayran, Nour Ali, and Roger Evans. 2016. A Systematic Mapping Study in Microservice Architecture. In *SOCA*. IEEE Computer Society, 44–51.

[4] A. Balalaie, A. Heydarnoori, and P. Jamshidi. 2016. Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture. *IEEE Software* 33, 3 (May 2016), 42–52. DOI: http://dx.doi.org/10.1109/MS.2016.64

[5] Len Bass, Ingo Weber, and Liming Zhu. 2015. *DevOps: A Software Architect's Perspective* (1st ed.). Addison-Wesley Professional.

[6] Keith H. Bennett and Václav T. Rajlich. 2000. Software Maintenance and Evolution: A Roadmap. In *Proceedings of the Conference on The Future of Software Engineering (ICSE '00)*. 73–87.

[7] Valerio Cosentino, Javier Luis Cánovas Izquierdo, and Jordi Cabot. 2015. Assessing the bus factor of Git repositories. In *22nd IEEE International Conference on Software Analysis, Evolution, and Reengineering, SANER 2015, Montreal, QC, Canada, March 2-6, 2015*. 499–503.

[8] Paolo Di Francesco, Ivano Malavolta, and Patricia Lago. 2017. Research on architecting microservices: Trends, focus, and potential for industrial adoption. In *Software Architecture (ICSA), International Conference on*. IEEE, 21–30.

[9] Robert M Emerson, Rachel I Fretz, and Linda L Shaw. 2001. Participant observation and fieldnotes. *Handbook of ethnography* (2001), 352–368.

[10] Daniel Escobar, Diana Cárdenas, Rolando Amarillo, Eddie Castro, Kelly Garcés, Carlos Parra, and Rubby Casallas. 2016. Towards the understanding and evolution of monolithic applications as microservices. In *Computing Conference (CLEI), 2016 XLII Latin American*. IEEE, 1–11.

[11] Eric Evans. 2003. *Domain-Driven Design: Tacking Complexity In the Heart of Software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[12] Jez Humble. 2018. Continuous Delivery Sounds Great, but Will It Work Here? *Commun. ACM* 61, 4 (March 2018), 34–39. DOI: http://dx.doi.org/10.1145/3173553

[13] Andrew J. Ko. 2017. A Three-year Participant Observation of Software Startup Software Evolution. In *Proceedings of the 39th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP '17)*. 3–12.

[14] Welder Luz, Gustavo Pinto, and Rodrigo Bonifácio. Building a Collaborative Culture: A Grounded Theory of Well Succeeded DevOps Adoption in Practice. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM 2018*.

[15] Robert Cecil Martin. 2003. *Agile Software Development: Principles, Patterns, and Practices*. Prentice Hall PTR, Upper Saddle River, NJ, USA.

[16] Genc Mazlami, Jürgen Cito, and Philipp Leitner. 2017. Extraction of Microservices from Monolithic Software Architectures. In *Web Services (ICWS), 2017 IEEE International Conference on*. IEEE, 524–531.

[17] Ronnie E. S. Santos, Fabio Q. B. da Silva, Maria Teresa Baldassarre, and Cleyton V. C. de Magalhães. 2017. Benefits and limitations of project-to-project job rotation in software organizations: A synthesis of evidence. *Information & Software Technology* 89 (2017), 78–96.

[18] Chris Stevenson and Andy Pols. 2004. An Agile Approach to a Legacy System. In *Extreme Programming and Agile Processes in Software Engineering*, Jutta Eckstein and Hubert Baumeister (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 123–129.

[19] Davide Taibi, Valentina Lenarduzzi, and Claus Pahl. 2017. Processes, Motivations, and Issues for Migrating to Microservices Architectures: An Empirical Investigation. *IEEE Cloud Computing* 4, 5 (2017), 22–32.

[20] Johannes Thones. 2015. Microservices. *IEEE Software* 32, 1 (2015), 116.

[21] Hulya Vural, Murat Koyuncu, and Sinem Guney. 2017. A Systematic Literature Review on Microservices. In *International Conference on Computational Science and Its Applications*. Springer, 203–217.

View publication stats