FACULDADE DE TECNOLOGIA DE SÃO PAULO

FELIPE MICHELON MARTINS

DIFICULDADES NO PROCESSO DE IMPLEMENTAÇÃO DE MICROSSERVIÇOS

SÃO PAULO

2021

FACULDADE DE TECNOLOGIA DE SÃO PAULO

FELIPE MICHELON MARTINS

DIFICULDADES NO PROCESSO DE IMPLEMENTAÇÃO DE MICROSSERVIÇOS

Trabalho submetido como exigência parcial para a obtenção do Grau de Tecnólogo em Análise e Desenvolvimento de Sistemas
Orientador: Professor Mestre Dionisio Gava Junior

SÃO PAULO

2021

FACULDADE DE TECNOLOGIA DE SÃO PAULO

FELIPE MICHELON MARTINS

DIFICULDADES NO PROCESSO DE IMPLEMENTAÇÃO DE MICROSSERVIÇOS

Trabalho submetido como exigência parcial para a obtenção do Grau de Tecnólogo em Análise e Desenvolvimento de Sistemas.

Parecer do Professor Orientador

_____

_____

_____

Conceito/Nota Final: _____

Atesto o conteúdo contido na postagem do ambiente TEAMS pelo aluno e assinada por mim para avaliação do TCC.

Orientador: Professor Mestre Dionisio Gava Junior

SÃO PAULO, ____ de _____ de 2021.

Assinatura do Orientador

Assinatura do aluno

RESUMO

O histórico de desenvolvimento de software teve uma tendência a criar grandes sistemas que continham todas as regras que são necessárias para cumprir os seus objetivos. Porém, uma nova técnica de arquitetura de software surgiu para evitar os problemas de ter uma base de código monolítica: os microsserviços. Desde o surgimento desta solução de arquitetura, muitos casos de sucesso mostraram que é algo que pode facilitar e melhorar o desenvolvimento de software para grandes propósitos e/ou com regras de negócio complexas, mas, apesar de parecer simples, não existe material de estudo para comprovar a real efetividade dos microsserviços em todos os tipos de empresas. Através de uma pesquisa qualitativa focada em um estudo de caso múltiplo, este trabalho busca expor a real eficácia desta técnica de arquitetura.

Palavras-chave: arquitetura de sistemas;desenvolvimento de software; microsserviços.

ABSTRACT

The historical background of software development had a bias to create large systems that held all rules necessary to accomplish its objectives. However, a new architecture technology emerged to avoid the problem of having a monolithic codebase: the microservices. Since the arrival of this architecture solution, many study cases have shown that it is something that can facilitate and improve software development with great purpose and/or with complex business rules, but, despite looking simple, there is no study material to endorse the real effectiveness of microservices on every kind of company. Through a qualitative research focused on a multiple case study, this work seeks to expose the real effectiveness of this architecture technique.

Key terms: systems architecture; software development; microsservices.

LISTA DE SIGLAS

UCP Unidade Central de Processamento

E2E End to end

HTTP Hypertext Transfer Protocol

IDE Integrated Development Environment

MVP Minimum viable product

REST Representational state transfer

SOA Service-Oriented Architecture

SOAP Simple Object Access Protocol

WSDL Web Services Description Language

SUMÁRIO

1. INTRODUÇÃO

Desde o começo dos anos 2000, o desenvolvimento de software distribuído ficou cada vez maior. Padrões de código e de arquitetura começaram a surgir e questionar os modelos da época, até que os microsserviços foram apresentados.

Com a premissa de mitigar problemas presentes em arquiteturas monolíticas e distribuídas, vários adeptos começaram a surgir no mercado e, com a influência de gigantes da tecnologia, os microsserviços cresceram exponencialmente. Entretanto, a adoção desse padrão pode gerar problemas nas organizações que não entendem a motivação e consequências da aplicação.

Portanto, este trabalho visa mapear e identificar quais problemas estão presentes na implementação de microsserviços, buscando casos reais, através de uma pesquisa qualitativa, para compará-los com os exemplos de riscos que a literatura expõe.

2. MICROSSERVIÇOS

O desenvolvimento tradicional de aplicações, desde os primeiros programas para computadores, são mais focados para a construção de aplicações monolíticas. Desse modo, a aplicação engloba todas as partes de negócio, levando a algumas desvantagens para sistemas muito grandes, como a dificuldade de solução de problemas e adição de novas funcionalidades. Para isso, os microsserviços fornecem uma arquitetura de solução que, além de incentivar o desenvolvimento, acelera a entrega de software.

Os microsserviços são tanto uma arquitetura, quanto uma abordagem para escrever sistemas. Com eles, as aplicações são divididas em componentes pequenos e independentes. Ao contrário da abordagem tradicional, que foca em construir uma única fonte de código que tem toda a responsabilidade, os microsserviços são componentes separados que trabalham juntos para realizar as mesmas tarefas.

Alguns autores ainda reforçam a definição de microsserviços como componentes que colaboram para um objetivo em conjunto: “Microsserviços são serviços pequenos e autônomos que trabalham juntos” (NEWMAN, 2015, p. 16) ou “arquitetura de microsserviços é uma abordagem para desenvolver uma aplicação, através de um grupo de pequenos serviços, cada um rodando no seu próprio processo” (FOWLER. 2014).

2.1. O tamanho ideal para um serviço

Uma grande questão é a real definição de “micro”, pois cada indivíduo pode ter um entendimento do tamanho ideal para um serviço. Isso ainda é um tema bem debatido na comunidade de programação.

Alguns autores defendem que o significado é mais voltado em como o serviço será desenvolvido:

> Um problema com o termo microsserviços é que a primeira coisa que você escuta é micro. Isso sugere que o serviço deve ser muito pequeno. [...] Na realidade, tamanho não é uma métrica útil.

> Um objetivo muito melhor é que um serviço bem definido é um serviço capaz de ser desenvolvido por times pequenos, com um pequeno tempo de entrega de forma mais independente (RICHARDSON. 2019, p. 43)

No entanto, a abordagem mais adotada é usando a opinião, de forma subjetiva, de quem está desenvolvendo e/ou mantendo o sistema:

> Quando estou falando em conferências, eu sempre faço um questionamento: quem tem um sistema muito grande e que gostaria de dividi-lo em outras partes? Quase todas as pessoas levantam a mão. Parece que nós temos um ótimo senso do que é muito grande, então pode-se dizer que, uma vez que um código não parece muito grande, provavelmente é pequeno o suficiente (NEWMAN. 2015, p. 17)

Portanto, não existe um consenso sobre o tamanho ideal de um microsserviço. O que pode-se extrair de cada autor, é que o grupo de pessoas, responsáveis pela base de código, podem definir qual o tamanho ideal dos microsserviços.

2.2. Histórico e necessidade

Aplicações são normalmente construídas em três partes: uma interface para o usuário, um banco de dados e uma aplicação que fica no servidor. A aplicação normalmente gerencia requisições *HTTP*, executa lógica de negócio, lê e escreve dados no banco de dados, e seleciona e popula os arquivos a serem enviados para o navegador do usuário. Essa aplicação é um monólito, ou seja, qualquer mudança no sistema, necessita de uma nova versão desta aplicação do servidor (LEWIS, FOWLER, 2014).

Monólitos podem podem ser bem sucedidos, mas o número de pessoas que se frustram com eles, especialmente com mais aplicações sendo implementadas na nuvem, cresce cada vez mais. Todas as mudanças estão unidas em um único ciclo, fazendo com que, com uma pequena alteração, todo o sistema tenha que ser reconstruído e implementado em seguida. Com o passar do tempo, fica mais difícil manter uma estrutura de módulos, deixando mais difícil com que as mudanças precisem mudar somente uma parte do sistema. A escalabilidade também fica mais

complexa, pois é necessário fazer isso com todo o sistema, ao invés de uma parte isolada que necessite de mais recursos (LEWIS, FOWLER, 2014).

Com o amadurecimento da arquitetura de monólitos, algumas técnicas de arquitetura foram criadas, como o *SOA*, que é uma solução de integração que se baseia muito em sistemas focados em produtos corporativos, com uma estrutura de serviços intermediários de grande porte. Porém, traz consigo outros recursos, além das práticas de desenvolvimento e manutenção de código, como o protocolo *SOAP* e os corpos *WSDL*, que não foram muito bem adotados pela comunidade de desenvolvimento (JAMSHIDI et al, 2018).

Em Maio de 2011, uma conferência sobre arquitetura de software citou o termo “microsserviços” pela primeira vez, dando nome a uma prática arquitetural que já estava sendo explorada pelos participantes. Alguns deles, como Werner Vogels, da Amazon, e Adrian Cockcroft, da Netflix, citaram termos como “encapsular as informações com uma interface de serviços com, cada um, operando com pequenas lógicas de negócio” e “arquitetura com contextos bem definidos e serviços pouco acoplados”, respectivamente. Outros conceitos citados partiram do contexto da época, como “*SOA* refinado” e “*SOA* feito corretamente” (JAMSHIDI et al, 2018).

Esses termos mostraram a necessidade de evoluir as técnicas da época, já que o *SOA* parecia interessante, mas não satisfazia as necessidades das pessoas. Um grande exemplo é a migração em massa de protocolos, de *SOAP* para *REST*, por ser mais leve e simples do que o antecessor. Outra grande necessidade, da época, era a facilidade para implementação de Design Orientado por Domínio, pois a facilidade que microsserviços traz para delimitação de contextos é muito maior do que os monólitos criados no *SOA*. Por fim, outras pequenas coisas, como design para falhas, isolamento de informações, automatização de estrutura, agilidade em escalabilidade e domínio do produto de ponta a ponta, eram mais fatores que eram requisitos para o desenvolvimento de software, e que eram facilitados pelos conceitos de microsserviços (JAMSHIDI et al, 2018).

2.3. Benefícios

2.3.1. Tecnologia heterogênea

Com um sistema composto de múltiplos serviços que colaboram entre si, é possível escolher qual tecnologia utilizar em cada um. Isso permite escolher as ferramentas certas para cada função desses serviços e não recorrer a uma solução padronizada, que é determinada pelo menor denominador comum das necessidades de todas as aplicações (NEWMAN, 2015, p. 4). Se uma parte do sistema precisa ser mais performática, ou precisa ser construída mais rapidamente, ou até mesmo precisa ter uma tecnologia que pessoas de negócio, com pouco conhecimento técnico, precisam dar manutenção, os microsserviços permitem que exista essa diferença de tecnologias, pois só é necessário que um padrão de comunicação seja estabelecido.

Também é possível adotar tecnologias novas e crescentes, de forma mais fácil. Caso seja interessante testar uma nova linguagem ou um framework, é possível fazê-lo sem apresentar muitos riscos, pois, no pior caso, esse microsserviço pode ser destruído e refeito usando algo mais confiável, sem comprometer o projeto inteiro (RICHARDSON. 2019, p. 17).

2.3.2. Isolamento de falhas

Como cada componente trabalha de forma totalmente isolada, o sistema fica mais resiliente e a prova de falhas, já que quando um serviço falha, os outros ainda continuam funcionando, pois um dos princípios da arquitetura de microsserviços é que cada componente deve continuar funcionando, mesmo quando uma ou mais dependências não estão disponíveis (INDRASIRI, SIRIWARDENA. 2018, p. 15)

Se algo parecido acontece com um sistema monolítico, um único erro pode derrubar toda a aplicação, afetando o sistema por completo. Por exemplo, se um componente do monólito apresenta vazamento de memória, isso pode comprometer todos os componentes da aplicação, mesmo que estejam funcionando perfeitamente (RICHARDSON. 2019, p. 16).

2.3.3. Escalabilidade

A crescente disponibilidade de infraestrutura em nuvem, configura um ambiente que facilita a configuração e manutenção de cada serviço (INDRASIRI, SIRIWARDENA. 2018, p. 15), permitindo que cada um deles seja clonado e/ou particionado de forma independente e sem afetar o sistema, além de prover um gerenciamento de custos muito mais detalhado e efetivo (NEWMAN, 2015, p. 6).

Isso é bem mais eficaz, se comparado a um monólito, que tem componentes com requisitos individuais e específicos, sendo provisionados ao mesmo tempo. Um exemplo, uma aplicação que um módulo que utiliza a UCP intensivamente, e outra que exige um grande espaço de memória, precisam ser provisionadas juntas, mesmo quando não estão sendo utilizadas ao máximo (RICHARDSON. 2019, p. 16).

2.3.4. Implementação independente

Com cada componente contido e isolado, é possível fazer mudanças e introduzir novas funcionalidades, sem comprometer o restante do sistema. Caso um problema ocorra, é mais fácil desfazer alguma alteração em um pequeno pedaço do sistema, do que desfazer uma grande mudança de um monólito (NEWMAN, 2015, p. 6).

2.3.5. Reaproveitamento de código

O isolamento de responsabilidades e domínios de cada serviço, provê um ambiente em que muitos sistemas podem aproveitar um comportamento, dentro da sua própria arquitetura. Quando um serviço disponibiliza uma funcionalidade, outros domínios podem utilizar o sistema como uma biblioteca, mas com o benefício de ter as regras do negócio inclusas.

Além disso, a comunicação dentro de uma rede fechada pode ser mais performática que utilizar algo externo. Requisições REST utilizando JSON , que são um padrão muito adotado no mercado, utilizam mais recursos do que uma rede interna se comunicando com uma lista de bytes.

2.4. Riscos

2.4.1. Complexidade de sistemas distribuídos

A natureza de sistemas distribuídos faz com que os microsserviços possam estar em máquinas diferentes, gerando latência na rede. Isso cria a necessidade de considerar. Isso implica que o tempo necessário para trocar informações entre sistemas deve ser considerado, pois aplicações que precisam obter respostas rapidamente são impactadas. Além disso, a rede também pode falhar por causa de perda de pacote, desconexão, problemas com segurança, etc (NEWMAN, 2020, p. 7).

IDE's e algumas ferramentas de desenvolvimento são, normalmente, focadas para o desenvolvimento de aplicações monolíticas, não provendo um suporte direto para sistemas distribuídos. Isso pode exigir que as pessoas desenvolvedoras utilizem técnicas mais avançadas e sofisticadas para o desenvolvimento integrado das aplicações (RICHARDSON. 2019, p. 16).

Outro grande problema é a automatização de testes integrados entre aplicações e/ou testes de ponta a ponta (*E2E*) sobre fluxos inteiros (RICHARDSON. 2019, p. 16). Determinar um contrato para que os testes sejam realistas e sempre estejam atualizados, gera alta necessidade de comunicação e convenções entre os times. Dependendo da forma como o código é mantido, cada serviço pode ter o seu repositório individual, aumentando ainda mais a complexidade para manter diversos contratos de uma fluxo *E2E*.

2.4.2. Domínio de tecnologias de implementação

Conteinerização e orquestração de sistemas são dois conceitos essenciais para a adoção de microsserviços. Caso a infraestrutura não suporte essas técnicas, será necessário investir em estudo para que os sistemas possam evoluir facilmente. Ambos os conceitos facilitam que novas aplicações sejam criadas, implementadas e gerenciadas de forma simples.

Rastreamento, métricas e *logs*, que são os pilares da observabilidade, também se fazem necessários para a utilização de uma arquitetura distribuída. Conseguir entender a cadeia de chamadas entre as aplicações, as métricas que elas geram e como cada uma se comporta, são informações que facilitam a manutenção e gerenciamento de uma rede de sistemas

Também é necessário um plano para implementar uma nova versão de um serviço essencial, já que muitos outros componentes podem depender dele (RICHARDSON. 2019, p. 16). Caso haja uma mudança que quebre algum contrato, todas as dependências já devem estar preparadas, ou isso pode interromper o funcionamento de fluxos inteiros.

2.4.3. Manutenção

A arquitetura de microsserviços normalmente é aplicada com diversas tecnologias e plataformas ascendentes no mercado, o que dificulta a integração entre elas (WOLFF, 2019). Se surgir a necessidade de integrar múltiplos serviços com uma única ferramenta, é necessário que toda a plataforma atual seja compatível, o que limita a gama de escolhas e futuras expansões do sistema.

Além da dificuldade de expansão técnica, também existe a limitação humana, pois o time responsável pelo gerenciamento de um grupo de serviços, precisa dominar grande parte, senão todas as ferramentas que compõem aquela parte do sistema. Isso pode gerar uma necessidade por profissionais com conhecimento específico, ou dedicar grande parte de recursos em treinamentos somente para a manutenção cotidiana das aplicações.

2.4.4. Escopos mal divididos

A primeira falha que pode acontecer com a arquitetura de microsserviços é a má divisão dos serviços. Como a divisão depende muito dos conceitos de Design Orientado por Domínio, a má separação pode resultar em problemas que se escalam para a empresa inteira. A lei de Conway diz que com o tempo, a estrutura de um software tenderá a replicar a estrutura de comunicação da empresa que o desenvolve, ou seja, se as aplicações forem separadas de um jeito que não reflita o

domínio da empresa, o desenvolvimento de software precisará levar um tempo para se adaptar ao modelo organizacional.

Mesmo com uma divisão adequada dos escopos dos serviços, a construção deles precisa ser feita de modo que tudo seja a prova de falha. Caso um ou poucos serviços apresentem problemas com isso, é possível que todo o sistema fique indisponível por causa disso (WOLFF, 2019).

2.4.5. Comunicação

O desenho da comunicação entre serviços deve ser robusto, pois programadores tendem a desenhar fluxos síncronos pela facilidade e costume com as ferramentas mais convenientes, como *REST*. Essa comunicação síncrona pode encadear um número muito grande de serviços, podendo gerar problemas de performance e indisponibilidade do sistema (WOLFF, 2019).

A solução para isso é a comunicação assíncrona, que também pode gerar outros tipos de problemas, pois é inaplicável em alguns cenários que a resposta precisa ser imediata. E este tipo de comunicação não afeta somente o meio de comunicação, mas a arquitetura como um todo, então é necessário que faça parte da concepção da ideia para funcionar sem imprevistos (WOLFF, 2019).

2.5. Aplicabilidade

Quando o negócio visa velocidade de crescimento, de forma fácil de segura, em várias áreas independentes e paralelas, os microsserviços são uma ótima forma de abordagem, como disse o diretor de tecnologia da Amazon:

> Agora, nós podemos construir aplicações complexas, a partir de serviços primitivos, que são relativamente simples. Nós podemos escalar a nossa operação de forma independente, mantendo a disponibilidade do sistema, e introduzir novos serviços rapidamente sem precisar de uma reconfiguração massiva (WERNWE. 2006, p. 16)

Focando em escalabilidade e independência de componentes, a Amazon conseguiu aumentar a velocidade de entrega, enquanto melhorou a segurança dos ambientes (NADAREISHVILI et al., 2016, p. 14).

Caso o objetivo esteja mais focado em inovação e testes de hipóteses, prontas para se tornarem um produto mínimo viável (*MVP*), o vice-presidente de engenharia da Gilt também enumera como o modelo pode auxiliar:

> A organização conseguiu os seguintes benefícios:
> - Menor dependência entre times, resultando em uma produção mais rápida de código;
> - Permite que muitas iniciativas ocorram em paralelo;
> - Promove a facilidade de inovação com “código descartável”, sendo fácil de continuar caso haja alguma falha (BRYANT, 2015).

Entretanto, não são todos os tipos de empresas que estão preparadas para esse tipo de arquitetura. Para que essa técnica funcione adequadamente, são necessários alguns pilares, como alinhamento da organização, possibilidade de implementação independente e domínio de múltiplas tecnologias. Essas características permitem que exista um ambiente seguro para criação, descarte e manutenção das aplicações (NADAREISHVILI et al., 2016, p. 15).

Quando o negócio começa a exigir que um ambiente complexo comece a se desenvolver rapidamente, os microsserviços também são uma solução. Entretanto, as fronteiras de negócio, tanto entre times, quanto entre serviços, devem estar muito bem definidas e maduras. Assim, não existirão conflitos ou falhas de negócio por problemas organizacionais (NADAREISHVILI et al., 2016, p. 15).

3. ESTUDO DE CASO MÚLTIPLO

Tendo como objetivo identificar quais problemas estão presentes na implementação de microsserviços e, baseado na literatura sobre o tema, verificar se o que está documentado na teoria acontece na prática, foi feito um estudo de caso múltiplo.

A pesquisa foi feita com duas empresas, com cenários diferentes, de forma qualitativa. Os dados sobre informações da empresa, tecnologias utilizadas e planejamento, foram colocados como texto aberto. As seções sobre motivação e problemas da implementação dos microsserviços foram colhidas utilizando um modelo semelhante à escala Likert, onde 0 significa “Discordo totalmente”, e 10 significa “Concordo totalmente”.

3.1. Resultados

3.1.1. Informações das empresas

Questão	Empresa A	Empresa B
Razão social	Creditas Soluções Financeiras LTDA.	CONTRATADO TECNOLOGIA LTDA
Ramo de atividade	Fintech	Recrutamento e Seleção
Tempo de atividade no mercado	8 anos	7 anos
Número de colaboradores	2000	150
Número de colaboradores em TI	450	30
Função do respondente	Engineering Lead	Engenheiro de Software

3.1.2. Detalhes sobre a implementação

Questão	Empresa A	Empresa B
Quais são as principais tecnologias utilizadas?	Kotlin + Spring + Java + Postgres + MongoDB + Ruby	Ruby + Rails + Postgres
Quais são as ferramentas utilizadas?	AWS, CircleCI, Docker, Insomnia, Kafka, Kibana, Kong, Kubernetes, Log stash, New Relic, OpenShift, Postman, Rollbar, Sentry, Swagger	AWS, Docker, Kibana, New Relic, Prometheus, Rollbar, RabbitMQ para comunicação entre os serviços, Coralogix para Logs, Grafana para visualização de observabilidade, PagerDuty para gerenciamento de incidentes, Github Actions para CI e Deploys
Quantas pessoas estão envolvidas no desenvolvimento e/ou manutenção dos microsserviços	150	4
Como foi o processo de implementação dos microsserviços?	Por módulos (mudança completa das partes do sistema, uma por vez)	Por módulos (mudança completa das partes do sistema, uma por vez)
Quais são as atividades sendo feitas atualmente para a implementação dos microsserviços?	Mapeamento de domínios e definição de times responsáveis	Separação do monólito em duas grandes aplicações. Depois disso, cada equipe planeja e executa uma extração ou criação de um módulo
Quais são as atividades planejadas para o futuro? (1-2 anos)	Extração de novos serviços, flexibilização da plataforma criação de ferramentas de infraestrutura e adoção de novas ferramentas de infraestrutura	Completar a extração das duas grandes aplicações do monólito

3.1.3. Motivação

Questão	Empresa A	Empresa B
Gostaria de ter tecnologias heterogêneas entre serviços	0	2
Gostaria isolar as falhas dos serviços	7	10
Gostaria escalar cada serviço individualmente	4	8
Gostaria reaproveitar o código	8	8
Gostaria de ter separação de responsabilidade de negócio entre serviços	10	10

3.1.4. Problemas durante a implementação

Questão	Empresa A	Empresa B
Houve problemas de rede entre os serviços	10	2
Houve problemas com testes entre serviços	4	10
Houve dificuldade na separação de domínios dos serviços	3	6
Houve conhecimento técnico necessário	5	8
Houve problemas com observabilidade	10	1
Houve quebra de contrato entre aplicações	3	6
Houve pessoas suficientes para dar manutenção aos sistemas	4	3
Houve limitação da expansão por necessidade de retrocompatibilidade	3	3
Houve má divisão dos escopos dos serviços	3	5

4. CONCLUSÃO

A crescente adoção dos microsserviços se mostra muito eficiente e eficaz contra diversos problemas já conhecidos na comunidade de software, fazendo uso de conceitos muito bem estabelecidos e aceitos, como sistemas distribuídos e Design Orientado por Domínio. Entretanto, esse padrão pode parecer uma “bala de prata”, quando uma única solução resolve todos os problemas sem efeitos colaterais, se não for estudado previamente e, se for adotado cegamente, mais problemas podem surgir.

A literatura sobre o tema descreve alguns dos problemas que podem acontecer com a adoção dessa técnica de arquitetura, porém, não existe um detalhamento explicando se os riscos surgem em empresas de um nicho específico, com um número determinado de colaboradores, ou se a equipe tem maturidade para adotar os microsserviços. Os riscos são genéricos e parecem ser aplicados a todos os casos, o que dificulta entender quais cenários são mais propícios para que dificuldades se manifestem.

A pesquisa qualitativa feita neste trabalho auxilia em explicar cenários que os problemas podem acontecer, unindo os riscos mapeados pelas pessoas influentes na academia, aos cenários de empresas diferentes que estão no processo de aplicação de microsserviços.

A empresa A tem uma grande parte dos colaboradores no departamento de desenvolvimento, isso pode ser uma das razões de estar no ramo de *Fintechs.* Entretanto, um dos grandes problemas reportados foi com a seção de observabilidade. Isso pode ser justificado pela falta de ferramentas utilizadas que poderiam auxiliar nesse aspecto. Se comparada a empresa B, que utiliza ferramentas propícias para resolver esse problema (Prometheus, Coralogix, Grafana), esse risco poderia ser mitigado mais facilmente. O fato de compreender de forma mediana como microsserviços devem ser implementados, também pode ser um fator de que as ferramentas de observabilidade podem estar mal configuradas e/ou sendo mal utilizadas.

Outra grande dor, da empresa A, é a comunicação entre serviços. Como a definição de times responsáveis ainda está ocorrendo, existe a oportunidade para que a interseção de dois riscos apareça: problemas de manutenção, por escopos mal divididos. Mesmo que a empresa não ache que isso seja um problema, também existe a necessidade de que cada aplicação esteja preparada para as falhas dos outros serviços, ou seja, mesmo que o domínio esteja separado corretamente, os sistemas também precisam estar separados corretamente. Se um sistema não tem um escopo bem definido, graças aos times ainda estarem se desenvolvendo, a manutenção desses sistemas também será precária, pois não existe grupo responsável por uma aplicação, deixando que problemas comecem a surgir sem acompanhamento, até que se mostrem como falha na comunicação.

Por outro lado, a empresa B reportou grande dificuldade com testes entre serviços. Isso se dá pela falta de ferramentas que poderiam auxiliar nisso, como PACT ou uma ferramenta de integração contínua, como CircleCI ou Jenkis. O grupo pequeno de pessoas envolvidas no desenvolvimento de microsserviços também é um grande fator para que isso aconteça, pois não é viável que somente quatro pessoas consigam dominar todos os pilares necessários para uma boa manutenção dos microsserviços.

A falta de conhecimento técnico necessário também é um grande limitador do avanço da separação do monólito, para uma arquitetura distribuída. Apesar de parte da motivação ser técnica, como reaproveitar o código, escalar cada serviço individualmente e isolar as falhas dos serviços, tudo ainda vai estar limitado à maturidade técnica dos times responsáveis. Compreender a complexidade de sistemas distribuídos e dominar as tecnologias de implementação são fatores essenciais para a evolução deste caso. Se o grupo responsável por essa implantação continuar pequeno e sem a evolução necessária, o número de problemas vai subir em pouco tempo.

Por fim, ficou aparente que os riscos não são tão óbvios como parecem, pois grandes empresas ainda podem ter problemas que pequenas não têm, ou vice-versa. As dificuldades da implementação de microsserviços é individual de cada

caso, pois os cenários de cada projeto dependem do ponto de partida, das equipes responsáveis, do objetivo a ser cumprido e das ferramentas disponíveis.

A possibilidade de realizar outra pesquisa, dessa vez quantitativa, poderia auxiliar no mapeamento e na identificação de padrões na das dificuldades da implementação dos microsserviços. A junção dos casos apresentados neste trabalho, com uma futura pesquisa exemplificando casos com os mesmos problemas, seria um grande avanço no cenário de microsserviços, pois deixaria mais visível como cada problema pode se manifestar em casos semelhantes.

5. REFERÊNCIAS

NEWMAN, Sam. **Building Microservices: Designing Fine-Grained Systems**. Sebastopol: O'Reilly Media, 2015.

RICHARDSON, Chris. **Microservices Patterns: With examples in Java**. Shelter Island: Manning, 2019.

FOWLER, Martin**.** Microservices. a definition of this new architectural term. **martinFowler.com**. 2014 Disponível em: <https://martinfowler.com/articles/microservices.html>. Acesso em: 01 set. 2020.

NEWMAN, Sam. **Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith**. Sebastopol: O'Reilly Media, 2020.

INDRASIRI, K.; SIRIWARDENA, P. **Microservices for the Enterprise: Designing, Developing, and Deploying**. San Jose: Apress, 2018.

JAMSHIDI, Pooyan et al. Microservices: The road so far and challenges ahead. **IEEE Software**, Pittsburgh, v. 35, n. 3, p. 24-35, maio-jun. 2018. Disponível em: <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8354433>. Acesso em: 24 set. 2020.

WOLFF. Eberhard. Why Microservices Fail: An Experience Report.In: International Conference on Microservices, 2019, Dortmund, **Paper...** University of Applied Sciences and Arts Dortmund, 2019.

WERNWE, Vogels. **A Conversation with Werner Vogels**. [Entrevista concedida a] Jim Gray. ACM Queue, vol. 4, n. 4, p. P 14-22, maio, 2006.

NADAREISHVILI, Irakli *et al*. **Microservice Architecture: Aligning Principles, Practices, and Culture**. Sebastopol: O'Reilly Media, 2016.

WERNWE, Vogels. **A Conversation with Werner Vogels**. [Entrevista concedida a] Jim Gray. ACM Queue, vol. 4, n. 4, p. P 14-22, maio, 2006.

BRYANT, Daniel. Scaling Microservices at Gilt with Scala, Docker and AWS. **InfoQ**. 2015. Disponível em: <https://www.infoq.com/news/2015/04/scaling-microservices-gilt>. Acesso em: 04 jan. 2021.

6. GLOSSÁRIO

Observabilidade: Uma medida que expõe o estado de um sistema, a partir de três itens: rastreamento, para identificar o caminho de uma informação entre serviços, métricas, para entender como as aplicações está se comportando, e logs, para exibir como cada serviço está lidando com as requisições.

Retrocompatibilidade: Característica, de novos sistemas, para que ainda respeitem as interfaces e comportamentos de aplicações legadas, que estão no mesmo sistema.