Universidade Federal de São Carlos – UFSCar

Departamento de Computação

Programa de Pós-Graduação

Alex Malmann Becker

O Impacto do Uso de Micro Serviços na Evolução de uma Linha de Produto de Software

São Carlos - SP

Agosto/2019

Alex Malmann Becker

O Impacto do Uso de Micro Serviços na Evolução de uma Linha de Produto de Software

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação, área de concentração: Engenharia de Software.

Orientador: Prof. Dr. Daniel Lucrédio

São Carlos - SP
Agosto/2019

UNIVERSIDADE FEDERAL DE SÃO CARLOS

Centro de Ciências Exatas e de Tecnologia
Programa de Pós-Graduação em Ciência da Computação

Folha de Aprovação

Assinaturas dos membros da comissão examinadora que avaliou e aprovou a Defesa de Dissertação de Mestrado do candidato Alex Malmann Becker, realizada em 20/08/2019:

Prof. Dr. Daniel Lucrédio
UFSCar

Prof. Dr. Elder de Macedo Rodrigues
UNIPAMPA

Profa. Dra. Rosana Teresinha Vaccare Braga
USP

Certifico que a defesa realizou-se com a participação à distância do(s) membro(s) Elder de Macedo Rodrigues, Rosana Teresinha Vaccare Braga e, depois das arguições e deliberações realizadas, o(s) participante(s) à distância está(ao) de acordo com o conteúdo do parecer da banca examinadora redigido neste relatório de defesa.

Prof. Dr. Daniel Lucrédio

Este trabalho é dedicado primeiramente a Deus,
por ser a base fundamental em minha vida, meu guia e amigo,
ao meu pai Rogério, minha mãe Lurdes, e aos meus irmãos Arthur, Evandro e Sophia,
e em especial ao meu irmão Everton, que me manda forças aonde quer que esteja.
Também dedico aos meus amigos e principalmente a minha noiva Pâmela,
que sempre me apoia e está ao meu lado em todos os momentos.

Agradecimentos

O agradecimento vai em especial ao meu orientador que me apoiou e incentivou durante todo o tempo, principalmente pelo voto de confiança que depositas-te em mim.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

“Alguns homens veem as coisas como são, e dizem ‘Por quê?’
Eu sonho com as coisas que nunca foram e digo ‘Por que não?’“
(Geroge Bernard Shaw)

Resumo

Com o avanço das tecnologias atuais e a alta demanda de software como serviço, cada vez mais as empresas estão se vendo obrigadas a modernizar suas aplicações para o mundo digital. No entanto, essa modernização precisa ser muito bem projetada para conseguir acompanhar as necessidades dos clientes e também manter um custo baixo de operação para a empresa, principalmente em aplicações que possuem uma grande variabilidade de funcionalidades para seus clientes. Nesse contexto, um dos problemas enfrentados é em relação ao gerenciamento de múltiplas versões de sistemas "semelhantes" e da evolução do software. Para resolver a questão do gerenciamento de múltiplas versões utiliza-se a abordagem de Linha de Produto de Software (SPL), enquanto que o estilo arquitetural de Micro Serviço é uma forma de conseguir deixar o processo de evolução mais leve, uma vez que é necessário realizar apenas a atualização do micro serviço específico. Todavia, apesar de ser um tema importante e bastante utilizado pela indústria de software e considerando que os benefícios da arquitetura de micro serviços são bem conhecidos em termos de operação e implantação, seu uso em linha de produtos não é objeto de muitos estudos na literatura, do ponto de vista da Engenharia de Software. No levantamento teórico realizado neste trabalho, foram encontrados apenas dois trabalhos que utilizaram micro serviços e SPL, sendo um deles, diretamente relacionado a este trabalho, pois também evoluiu uma aplicação utilizando o estilo arquitetural de micro serviços e um processo de SPL, e o outro que utilizou micro serviços e SPL para melhorar a flexibilidade da variabilidade da família do produto. Entretanto, os artigos abordam de forma resumida o processo de como foram projetados os serviços, e com isso este ponto precisa de mais investigação. Além disso, o principal foco da avaliação foi em tarefas de manutenção, e não evolução, como investigado nesta pesquisa. Portanto, este trabalho teve como objetivo avaliar a evolução (tanto na adição de novas *features*, como na manutenção de *features* existentes) de uma SPL, através da utilização de micro serviços, para no final poder afirmar se o impacto de micro serviços realmente melhorou a tarefa de evolução. Para conseguir chegar ao objetivo esperado, este trabalho teve a parceria de uma empresa sediada no Rio Grande do Sul e que possui em seu portfólio de produtos um Sistema de Gestão Empresarial (ERP), o qual é implementado em uma linha de produto. Com base nisso, foi executado um estudo de caso em dois cenários (um com micro serviço e outro sem micro serviço), o que nos permitiu avaliar que a utilização de micro serviços traz grandes benefícios na evolução de uma linha de produto de software, principalmente nas tarefas de manutenção adaptativas. Por fim, além da avaliação do impacto do uso de micro serviços na evolução de uma linha de produto, este trabalho contribui com as lições aprendidas da migração de um projeto desenvolvido na abordagem tradicional para o uso de micro serviços, utilizando dados de um experimento real com foco na Engenharia de Software.

Palavras-chaves: SPL, Micro Serviços, Evolução.

Abstract

Due to the high demand for software as a service, more and more companies are forced to modernize their applications to the digital world. However, this modernization needs to be very well designed to be able to follow the customers needs and also to have low operation cost, especially in applications that have a great deal of functionality variability for their customers. In this context, one of the faced problems is related to the management of multiple versions of "similar" systems and software evolution. To solve this issue, the SPL approach is used, while the Micro Services architectural style is a way to make the evolution process lighter, since it can be performed only in the specific micro-services being updated. However, despite being an important subject and widely used by the software industry, in the academy we did not find a large number of papers dealing with micro services and SPL. In the survey carried out in this work, only two papers were found that used micro services and SPL, one of them directly related to this work, applying the micro-services architecture and a SPL process, and the other work which used micro services and SPL to improve the flexibility of product family variability. However, the articles briefly discuss the process of how services were designed, and so this point needs more research. In addition, the main focus of the evaluation was on maintenance tasks, not evolution, as we investigated in this research. Therefore, this work aims to evaluate the evolution (both in the addition of new features and in the maintenance of existing features) of a SPL through the use of micro services in order to be able to affirm if the impact of micro services really improved the task of evolution. In order to reach the expected objective, this work had the partnership of a company based in Rio Grande do Sul, which has a ERP in its product portfolio, implemented as a product line. Based on this, a case study was carried out in two scenarios (one with micro services and the other without micro services), which allowed us to evaluate that the use of micro services brings great benefits in the evolution of a software product line, mainly in adaptive maintenance tasks. Finally, in addition to evaluating the impact of the use of micro services in the evolution of a product line, this work contributes with the lessons learned from the migration of a project developed in the traditional approach to the use of micro services, using data from a real experiment with focus on software engineering.

Key-words: SPL, Microservices, Evolution.

Acrónimos

ABS Abstract Behavioral Specification

API Application Program Interface

CVL Linguagem Comum de Variabilidade

DFe Documento Fiscal Eletrônico

EAR Enterprise Application aRchive

ERP Sistema de Gestão Empresarial

GQM Goal-Question-Metric

HTTP Hypertext Transfer Protocol

JPA Java Persistence API

JSF JavaServer Faces

M2M Model-To-Model

MVC Model–View–Controller

PLA Arquitetura de Linha de Produto

SaaS Software como Serviço

SOA Service-Oriented Architecture

SPL Linha de Produto de Software

SPLE Engenharia de Linha de Produto de Software

WAR Web application ARchive

Sumário

1 Introdução 17

2 Fundamentação Teórica 21

2.1 Linha de Produto de Software (SPL) 21

2.2 Micro Serviços 24

2.3 Considerações Finais 27

3 Revisão da Literatura e o Estado da Arte 29

3.1 Metodologia de Pesquisa 29

3.2 Trabalhos Relacionados 29

3.2.1 Técnicas de SPL e Micro Serviços 30

3.2.2 Outros Trabalhos Relacionados 32

3.3 Considerações Finais 37

4 O Impacto do Uso de Micro Serviços na Evolução de uma Linha de Produto de Software 39

4.1 Migração para Arquitetura de Micro Serviços 43

4.2 Estudo de Caso 46

4.2.1 Objetivos 47

4.2.2 Seleção de contexto, variáveis e hipóteses 48

4.2.3 Planejamento e execução 49

4.2.4 Resultados e discussão 52

4.2.5 Ameaças à validade 57

4.3 Considerações Finais 59

5 Considerações Finais e Trabalhos Futuros 61

A Casos de Uso Utilizados no Estudo de Caso 65

B Casos de Teste Utilizados no Estudo de Caso 79

Referências 85

1 Introdução

Uma Linha de Produto de Software (SPL) é um conjunto de sistemas que compartilham um mesmo conjunto de características (*features*) e que satisfazem as necessidades de um segmento específico de mercado (COHEN, 2002). Esses conjuntos de sistemas pertencem a uma mesma família de produtos e estão relacionados a partir de artefatos similares e projetados sobre uma arquitetura comum. Atualmente, muitas empresas utilizam essa abordagem de desenvolvimento, que é uma abordagem que muda o desenvolvimento e gerenciamento de um único sistema para a gestão de famílias de sistemas. Para isso, a maioria das empresas utilizam ferramentas de apoio para automatizar esse processo, principalmente na gestão da variabilidade da SPL, configuração do produto e modelagem do domínio. Entretanto, algumas empresas, apesar de não utilizarem um controle mais rigoroso da gestão da SPL devido ao seu alto custo de implantação, acabam gerenciando suas linhas de forma *ad-hoc*, ou seja, sem a utilização de ferramentas para automatizar o uso de SPL, utilizando planilhas e tabelas para auxiliar nesse processo (STOIBER; GLINZ, 2010). Além disso, a implementação do controle de seus produtos é geralmente realizada por exemplo, com diretivas de compilação condicional ou até mesmo no caso de múltiplos clientes, compilando seus produtos para cada cliente específico, o que acaba gerando um custo e prazo de evolução elevados.

Além disso, com o avanço das tecnologias atuais e da alta demanda de software como serviço, cada vez mais as empresas estão se vendo obrigadas a modernizar suas aplicações para não perder espaço no competitivo mercado de negócio, pois os usuários desejam poder acessar as informações do sistema a qualquer lugar e hora, e não possuir essa funcionalidade é sinônimo de que a empresa logo perderá seus clientes. Essa modernização e consequentemente a migração para Software como Serviço (SaaS) trazem muitos benefícios, tanto para a empresa como para seus usuários, como por exemplo, um melhor controle da evolução e entrega rápida das funcionalidades do software, baixo custo, e diminuição da necessidade de manutenção da infraestrutura física de redes locais cliente/servidor. Por outro lado, o usuário terá acesso a qualquer dispositivo conectado à Internet, acesso ao sistema a qualquer dia e horário, além de pagar apenas pelo que for utilizar do sistema, entre outros benefícios. Vale ressaltar que SaaS é um modelo de negócios, de entrega de software - o software é utilizado como serviço. Isso não implica que sua arquitetura interna seja baseada em serviços, apesar de isso ser muito comum.

Também, diante dessa modernização para sistemas SaaS, um estilo arquitetural que vem ganhando forças nos últimos anos na indústria é o de micro serviços, que fornece ainda mais benefícios para a empresa, como por exemplo, a fácil e rápida implantação de novos serviços (*build* e *deploy*), em poder possuir times especialistas em diversos pontos

ou regras de negócios e implementados em diferentes linguagens (MOREIRA; BEDER, 2016). Nesse sentido, Newman (2015), Lewis e Fowler (2014) definiram alguns princípios que orientam os engenheiros de software a projetarem seus micro serviços, pois ainda não existe um consenso pela comunidade do tamanho de um micro serviço.

Exemplos de grandes empresas que já passaram por esse processo de migração de suas aplicações para o uso de micro serviços são a Amazon e Netflix (THÖNES, 2015). No entanto, essa modernização precisa ser muito bem projetada para conseguir acompanhar as necessidades dos clientes e também manter um custo baixo de operação para a empresa, principalmente em aplicações que possuem uma grande variabilidade de funcionalidades para seus clientes.

Nesse contexto, um dos problemas da migração de software voltado para múltiplos clientes e com uma grande personalização de suas funcionalidades é conseguir gerenciar e projetar desde o início toda a variabilidade da aplicação, pois dependendo de como essa variabilidade é projetada, pode tornar inviável a manutenção ou até mesmo gerar um alto custo de manutenção para readequar a solução. Para resolver isso, um dos meios é a utilização da abordagem de SPL que facilita o desenvolvimento de soluções voltadas para linha de produto, principalmente através da reutilização de código e gerenciamento de versões.

Além disso, outro problema enfrentado é em relação ao gerenciamento de múltiplas versões de sistemas parecidos e a evolução do software. Existem diversas estratégias utilizadas pela indústria para projetar suas aplicações na nuvem. Uma delas é a escolha de fazer o *deploy* de suas aplicações web repetindo o arquivo de *deploy* para cada cliente (por exemplo, no caso do Java o arquivo Web application ARchive (WAR) ou Enterprise Application aRchive (EAR)). Outra estratégia que pode ser adotada é a utilização de um único arquivo de *deploy* para todos os clientes, e nesse caso, é necessário uma programação da aplicação para conseguir gerenciar o que cada cliente utilizará. Em ambas as abordagens, é necessário um processo bem definido no controle de versões, e, também o auxílio de ferramentas para automatizar esse processo. Na primeira abordagem, com a necessidade da alteração de uma *feature* existente ou a adição de novas *features* far-se-á necessária a atualização em cada aplicação dos clientes, gerando um alto custo para empresa e um controle mais rigoroso desse processo, ou seja, considerando uma linha de produto com cinco clientes diferentes e é feito uma mudança ou alguma correção de *bug* em três deles, tornando-se necessário replicar essa mudança em todos os envolvidos. Enquanto que na segunda abordagem, além da aplicação ficar com um tamanho consideravelmente grande, o que elevará e muito o tempo de *deploy*, a cada atualização todos os clientes ficarão sem poder utilizar o sistema, levando grandes prejuízos aos evolvidos.

Nesse sentido, o estilo arquitetural de Micro Serviços é uma forma de conseguir deixar esse processo de evolução mais leve, uma vez que é necessário realizar apenas a

atualização do micro serviço específico e o restante da aplicação (no exemplo do Java, os arquivos WAR, que possuirão somente a camada de visualização) dos clientes continuará funcionando normalmente.

Todavia, apesar de ser um tema importante e bastante utilizado pela indústria de software, e considerando que os benefícios da arquitetura de micro serviços são bem conhecidos em termos de operação e implantação, seu uso em linha de produtos não é objeto de muitos estudos na literatura, do ponto de vista da Engenharia de Software. No revisão da literatura executada neste trabalho, foram encontrados apenas dois trabalhos que utilizaram micro serviços e SPL. Um deles, o trabalho de Tizzei et al. (2017), diretamente relacionado a este trabalho, pois também evoluiu uma aplicação utilizando o estilo arquitetural de micro serviços e um processo de SPL. E o trabalho de Naily et al. (2017) que utilizou micro serviços e SPL para melhorar a flexibilidade da variabilidade da família do produto. Porém ainda é necessário uma melhor avaliação da abordagem proposta pelos autores. Além disso, os micro serviços projetados fugiram dos princípios de micro serviços propostos por Newman (2015), Lewis e Fowler (2014).

Existem também outros trabalhos relacionados mas que não estão diretamente relacionados a este, pois esses trabalhos utilizaram ou SPL e/ou Service-Oriented Architecture (SOA) para avaliar a evolução de sistemas. É o caso do trabalho de Capilla e Topaloglu (2005), no qual os autores tentaram melhorar o desenvolvimento e a evolução de sistemas orientados a serviços, propondo uma modificação do processo tradicional de Engenharia de Linha de Produto de Software (SPLE). Também pode ser citado o trabalho de Horcas, Pinto e Fuentes (2016), onde se apresenta uma abordagem de Arquitetura de Linha de Produto (PLA) para gerenciar a variabilidade e a evolução de aplicações *multi-tenancy*[1] em nuvem, através de modelos de variabilidades baseados na Linguagem Comum de Variabilidade (CVL).

Através da revisão da literatura realizada, pode-se concluir que SPL e SOA já estão bastante consolidados tanto na indústria, como na academia, porém, apesar da abordagem de micro serviços também estar bastante consolidada na indústria, ainda carece de trabalhos que abordem micro serviços na academia, principalmente relacionados com SPL, o que mostra que esta pesquisa de mestrado beneficiará tanto a academia como a indústria, em relação a SPL e micro serviços na evolução de sistemas. Portanto, este trabalho teve como objetivo avaliar a evolução (tanto na adição de novas *features*, como na manutenção de *features* existentes) de uma SPL, através da utilização de micro serviços, para no final poder afirmar se o impacto de micro serviços realmente melhorou a tarefa de evolução. Este objetivo está de acordo com a sugestão de trabalhos futuros feita por Tizzei et al. (2017), que citaram a avaliação do uso de micro serviços na perspectiva da evolução do

[1] Multi-tenancy ou multi-inquilino é a capacidade de uma única instância de software fornecer seu serviço a várias partes/clientes simultaneamente (CALERO et al., 2010).

software, sendo esta a motivação do presente trabalho.

Para conseguir chegar ao objetivo esperado, este trabalho teve a parceria de uma empresa sediada no Rio Grande do Sul que desenvolve soluções empresariais e que possui um Sistema de Gestão Empresarial (ERP) com clientes em produção. Com base nisso, foi executado um estudo de caso em dois cenários distintos, com a intenção de conseguir medir o real impacto do uso de micro serviços. O primeiro cenário, que é o cenário atual do sistema da empresa, é uma aplicação java web que não utiliza a abordagem de micro serviços e cujo gerenciamento da linha de produtos é feito de maneira *ad-hoc*, ou seja, não utiliza recursos para auxiliar nesse processo, como por exemplo, o uso da modelagem de *features*. E no segundo cenário, foi a aplicação que utiliza a abordagem de SPL e micro serviços. Vale destacar que o cenário dois foi implementado pelo autor deste trabalho, seguindo as diretrizes propostas por Newman (2015) e Lewis e Fowler (2014).

Para realizar o planejamento deste estudo de caso foi levado em consideração o processo estabelecido por Wohlin et al. (2012). Nesse experimento, foram realizadas oito sessões de uma hora e meia com dois desenvolvedores, realizando quatro tarefas de evolução nos dois cenários. Também, a fim de realizar nossa base de comparação entre a evolução dos dois cenários, foi utilizada a técnica Goal-Question-Metric (GQM), onde foram estabelecidas as métricas, como por exemplo, número de linhas de códigos modificadas, número de arquivos modificados, número de horas para realizar a mudança, além das métricas dificuldade de testar e dificuldade para entregar o software, que foram avaliadas através de um questionário respondido pelos desenvolvedores após o término do experimento.

Observou-se que micro serviços melhoraram a evolução em tarefas adaptativas. Em um tipo de tarefa evolutiva, que são as tarefas de adicionar novas *features*, foram necessários mais trechos de código para realizar a evolução, exigindo portanto mais esforço. Em compensação, as tarefas adaptativas, menos código e esforço foi necessário com o uso de micro serviços.

Por fim, além da avaliação do impacto do uso de micro serviços na evolução de uma linha de produto, este trabalho contribui com as lições aprendidas da migração de um projeto desenvolvido na abordagem tradicional para o uso de micro serviços, utilizando dados de um experimento real com foco na Engenharia de Software.

2 Fundamentação Teórica

2.1 Linha de Produto de Software (SPL)

Linha de Produto de Software (em inglês, *Software Product Line*) é um modelo de processo que tem como principal estratégia a utilização do reúso de forma sistemática no desenvolvimento de sistemas, que pertencem a uma mesma família de sistemas. Entende-se por família de sistemas as aplicações que possuem um conjunto de funcionalidades em comum e que pertencem a um mesmo domínio de negócio. Diante das diversas definições de Linha de Produto de Software (SPL) que são encontradas na literatura, Cohen (2002) menciona que a principal definição que tem a maior aceitação na indústria é de Clements e Northrop (2002):

> "Uma linha de produto de software é um conjunto de sistemas que usam software intensivamente, compartilhando um conjunto de características comuns e gerenciadas, que satisfazem as necessidades de um segmento particular de mercado ou missão, e que são desenvolvidos a partir de um conjunto comum de ativos principais e de uma forma preestabelecida." (CLEMENTS; NORTHROP, 2002, p. 563).

A utilização desse modelo faz com que o processo de desenvolvimento de sistemas seja um pouco diferente da abordagem tradicional. Nessa abordagem, como mencionado anteriormente, o reúso está em maior enfoque. Também pode-se destacar a implementação específica dos clientes, pois essa abordagem permite projetar as variabilidades específicas de cada um, o que levará a uma maior satisfação dos clientes. Também, a partir do reúso, a organização passa a ter mais produtividade no desenvolvimento, um produto com mais qualidade e confiabilidade, além de, em tese, ter um custo menor e um tempo menor para entrega do produto ao cliente.

Cohen (2003) classificou as vantagens de SPL em dois grupos:

- **Tangível:** Nesse grupo, estão as vantagens que podem ser medidas de uma forma direta, como por exemplo, a redução de defeitos. Entre os benefícios tangíveis estão: lucratividade, uma vez que são criados produtos específicos de um dado segmento de mercado, levando a um aumento da lucratividade da organização; qualidade, pois no desenvolvimento desse tipo de sistema existe uma redução no número de defeitos relatados; performance dos produtos de software, através da maturidade da linha e da otimização constante de seus ativos; tempo de integração, pois nessa abordagem o desenvolvimento é incremental, e com isso, o tempo de integração é facilitado; e, por último, produtividade, a qual além de reduzir o custo de desenvolvimento,

pode levar a uma redução no tamanho da equipe, bem como no cronograma de lançamento da linha.

- **Intangível:** Nesse grupo, foram enquadradas as vantagens que não conseguem ser medidas com métricas e que são relatadas pela equipe de desenvolvimento, como por exemplo, a satisfação do cliente. Entre os benefícios intangíveis estão: menor desgaste de profissionais; aceitabilidade dos desenvolvedores, pois após um treinamento inicial, os desenvolvedores geralmente apoiam o uso dessa abordagem; satisfação profissional, pois os desenvolvedores tendem a trabalhar mais no aperfeiçoamento e/ou inovação do que na criação do zero dos recursos; e por último, a satisfação do cliente, pois tende-se a diminuir o tempo de entrega, reduzir os custos, além de diminuir a taxa de defeitos.

Um ponto importante que deve ser levado em consideração ao migrar um sistema para SPL é que podem ocorrer fatores não tecnológicos, como por exemplo, adaptabilidade das pessoas, pois SPL é um modelo que interfere na maneira da empresa trabalhar (DURSCKI et al., 2004). Além disso, o processo de SPL pode sofrer algumas resistências em sua implantação, e com isso, Cohen (2003) fez um levantamento com alguns dos principais problemas decorrentes do uso de SPL, que são:

- **Falta de um líder comprometido:** O líder deve ser uma pessoa que acredita nos princípios do modelo e que estará supervisionando o processo e motivando as pessoas, e caso algum desses pontos falharem, todo o processo corre risco de insucesso.

- **Falta de compromisso da gerência:** A gerência deve estar totalmente engajada e dispor de todos os recursos necessários, uma vez que trará grandes benefícios para a organização, pois a desistência desse processo trará além de prejuízos, funcionários desmotivados com a falha do projeto.

- **Falta de compromisso da equipe:** O líder deve criar estratégias para envolver toda a equipe no processo, pois uma equipe desacreditada com o modelo impossibilita todo o processo.

- **Interação insuficiente entre as equipes:** Como é um processo que envolve diversos setores da empresa, a falta de interação e colaboração entre essas equipes pode prejudicar a obtenção dos objetivos esperados pela organização.

- **Evolução da abordagem (melhoria contínua):** é necessário possuir um processo de melhoria contínua para que as práticas utilizadas não se tornem obsoletas e inapropriadas para evolução da linha.

Quanto ao desenvolvimento de uma linha de produto, existe um processo de desenvolvimento o qual é o mais utilizado na indústria de software comercial, composto por duas etapas principais. A primeira etapa, chamada de engenharia do domínio, é responsável por analisar a variabilidade dos requisitos e criar artefatos reutilizáveis. Ela é dividida na análise do domínio, que analisa o escopo e todas as possibilidades do domínio e tem como saída um conjunto de *features*, e na implementação do domínio, que com base na análise do domínio, implementa os artefatos reutilizáveis, gerando o código fonte. A segunda etapa, chamada de engenharia da aplicação, é responsável por reutilizar os artefatos e se divide em análise de requisitos e derivação do produto. Na análise de requisitos são selecionadas as *features* com base nas necessidades de cada cliente, enquanto na derivação do produto é realizado o processo para compor o produto conforme as *features* selecionadas na etapa anterior.

Outro ponto importante quando se trabalha com SPL é a gestão da variabilidade do software. É o ponto chave do sucesso de uma linha de produto, pois está diretamente relacionada com a evolução e customização da linha. Nessa gestão, são utilizados como base artefatos de referência, que auxiliam no desenvolvimento de novos produtos da família. Alguns desses artefatos são os requisitos de linha de produto, testes de linha de produto e arquitetura de linha de produto. A arquitetura de linha de produto é um dos principais artefatos gerados, pois é nela que é feito o vínculo entre as necessidades do cliente, do domínio e do novo produto. Segundo Pereira, Figueiredo e Costa (2015), o padrão para representar a variabilidade de uma SPL e todas as configurações dos produtos de uma família é o modelo de *features* (*feature model*), podendo ser usados alguns perfis UML (Linguagem de Modelagem Unificada) para isso. O modelo de *features* também tem um papel fundamental no processo da SPL, pois é ele que irá fornecer uma visão das semelhanças e pontos de variabilidades de uma família de sistemas. Esse modelo é composto basicamente pelo diagrama de características, que permite a visualização de recursos hierárquicos de uma SPL e suas relações, pelas regras de composição, que validam as combinações de *features* possíveis, e também pela análise relacional, que realiza recomendações para indicar o melhor cenário para uma *feature* ser ou não selecionada.

Por fim, a engenharia de SPL é responsável por definir processos, metodologias e outros recursos quaisquer que venham a facilitar o desenvolvimento de uma SPL, sempre visando à redução de custos, qualidade e o menor tempo de lançamento de uma linha de produto. Vale destacar que a utilização de ferramentas que automatizam o processo de SPL é de suma importância, principalmente no que diz respeito ao gerenciamento de variabilidades, pois com o passar do tempo, se não tiver o suporte de alguma ferramenta, pode ficar complicada a gestão de uma linha de produto.

2.2 Micro Serviços

Segundo Lewis e Fowler (2014) uma arquitetura de micro serviços é uma abordagem de desenvolvimento onde uma única aplicação é composta de um conjunto de pequenos serviços. Cada serviço possui seu próprio processo ou servidor dedicado, deve ser independente e escalável, é projetado conforme as regras de negócios, e a comunicação entre os serviços ocorre através de mecanismos leves, como por exemplo, por uma *Application Program Interface (API)*, e *Hypertext Transfer Protocol (HTTP)*.

Tanto na academia como na indústria ainda não existe um consenso do tamanho que um micro serviço deve possuir e muito é utilizado da experiência dos membros das equipes para definir a granularidade de cada serviço. Entretanto, Newman (2015) e Lewis e Fowler (2014) definem alguns princípios que orientam os engenheiros de software a projetarem seus micro serviços, que são:

- **Modelo em torno de conceitos de negócios:** Os micro serviços devem ser modelados em torno de contextos limitados, ou seja, em relação a regras de negócios.
- **Ocultar detalhes de implementação internas:** Para garantir a independência dos micro serviços, devem ser ocultos detalhes de implementação e banco de dados, e a comunicação entre os serviços deve ser realizada através de alguma API. Essa estratégia é um ponto chave para se obter mudanças futuras na arquitetura.
- **Descentralização:** Os micro serviços devem ser altamente coesos e com baixo acoplamento, de modo que modificações não afetem outros serviços.
- **Implantação independente:** Esse princípio estabelece que qualquer micro serviço deve poder ser lançado em produção sem afetar ou ter que implantar outros micro serviços. Essa estratégia aumenta a velocidade de lançamento de novos recursos e dá uma maior autonomia para as equipes de desenvolvimento.
- **Cultura de automação:** Devido à complexidade que os micro serviços podem adicionar ao aumentar o número de serviços independentes do sistema, sugere-se que a empresa adote a cultura de automação, utilizando processos e ferramentas de entrega contínua, por exemplo.
- **Isolar falhas:** Os micro serviços devem ser projetados para tolerar as falhas do serviço. Caso ocorra alguma falha, a aplicação deve responder da melhor maneira ao usuário final.
- **Altamente observável:** Apesar dos benefícios em quebrar um sistema em serviços com granularidade fina, isso acaba gerando uma complexidade alta no monitoramento do comportamento de múltiplos serviços. Com isso, sugere-se utilizar técnicas

como monitoramento semântico, que possibilita o monitoramento automático dos micro serviços.

Para entender melhor a arquitetura de micro serviços, primeiro é necessário entender o conceito de arquitetura monolítica. A principal característica da arquitetura monolítica é que ela possui uma única aplicação responsável por todas as tarefas do sistema, além de ser independente de outras aplicações. Além disso, nessa arquitetura, a aplicação é projetada sem modularidade externa, ou seja, não é projetada para ser base de outras aplicações. Em termos de modularidade, pode ser feita uma modularização interna, na qual será feito o link dos módulos na mesma aplicação e por fim, é necessário uma compilação de todos os módulos para gerar uma única aplicação.

Como o uso de qualquer tecnologia em desenvolvimento de software, existem vantagens e desvantagens em relação às aplicações monolíticas. Além disso, a utilização da arquitetura de micro serviço deve ser bem pensada no início do projeto, pois não é a solução para todos os cenários. Moreira e Beder (2016) citam que as principais vantagens do uso de micro serviços estão relacionadas com a heterogeneidade tecnológica, resiliência, escalabilidade e facilidade de implantação. A heterogeneidade tecnológica é uma vantagem pois os micro serviços, apesar de poderem ser desenvolvidos em linguagens diferentes, são autônomos e trabalham em harmonia, parecendo que todos utilizam a mesma tecnologia. Essa característica também permite que testes de novas tecnologias possam ser executados de uma forma rápida e com custo baixo, pois caso algo não dê certo basta apenas reescrever uma parte pequena do serviço específico. A resiliência é outra característica que traz grandes benefícios ao negócio. Dado que a resiliência é capacidade de um sistema se recuperar de falhas, em aplicações monolíticas caso ocorra alguma falha, todo o serviço da aplicação ficará indisponível, enquanto que com o uso de micro serviços, caso ocorra algum problema, apenas o serviço em questão irá ficar indisponível, ao passo que os outros serviços continuarão funcionando normalmente. Em relação à escalabilidade, com uso de micro serviços é possível escalar apenas os serviços específicos conforme a demanda do cliente. Em contrapartida, em aplicações monolíticas é necessário escalar toda a aplicação. A última característica é a facilidade de implantação, o que permite fazer o *deploy* apenas das *features* desejadas, e assim ter um controle mais fácil desse processo.

Moreira e Beder (2016) também citam as principais desvantagens do uso de micro serviços, que são a complexidade de desenvolvimento, chamadas remotas e gerenciamento de múltiplos bancos de dados e transações. A principal característica e um receio do uso de micro serviços perante a equipe de desenvolvimento é com o aumento da complexidade de desenvolvimento. Nas aplicações monolíticas, o processo de desenvolvimento é mais fácil de ser gerenciado, pois tudo está integrado em um único projeto, bastando apenas realizar importações de pacotes e classes. Já no desenvolvimento com micro serviços, é necessário possuir uma boa gerência de dependências entre os módulos dos micro serviços

e também uma boa comunicação entre a equipe, para que não haja problemas na troca de informações e problemas de versões desatualizadas. Outro ponto que leva uma certa desvantagem no uso de micro serviços são as chamadas remotas, pois a comunicação externa entre os micro serviços é geralmente mais custosa do que uma comunicação de classes internas, que existem em aplicações monolíticas. Por último, o gerenciamento de múltiplos bancos de dados e transações é outra característica importante, pois esse é um processo complexo e custoso uma vez que é necessário realizar uma série de tratamentos contra falhas dos serviços, para que assim se consiga ter um controle total das transações e dos dados.

Dragoni et al. (2017) mencionam que uma das principais características dessa abordagem é a escalabilidade, pois além dos benefícios que ela fornece em relação a desempenho, ela auxilia para garantir disponibilidade e tolerância a falhas. Com isso, aborda-se a seguir os principais tópicos relacionados para atingir a escalabilidade:

- **Distribuição:** Devido a arquitetura de serviços já serem distribuídos, a caraterística de distribuição não é intrínseca ao uso de micro serviços. O uso de micro serviços leva essa característica ao extremo, pois os serviços são projetados em tamanho relativamente pequeno e eles são independentes, o que possibilita implantar serviços em servidores diferentes, tornando o sistema mais eficiente do que o monolítico, uma vez que a aplicação terá uma escalabilidade maior diante da distribuição.

- **Portabilidade:** Como os micro serviços são implantados em *containers* que possuem todo o ambiente (bibliotecas, banco de dados, etc.) necessário para funcionar independente da plataforma, isso permite uma replicação sem esforço dos serviços individuais em plataformas heterogêneas.

- **Elasticidade:** A facilidade em replicar micro serviços individuais permite que as arquiteturas de micro serviço sejam elásticas, ou seja, é possível escalar dinamicamente os serviços individuais de acordo com a necessidade da carga de uso em determinados períodos, conforme a demanda de cada cliente.

- **Disponibilidade:** Devido aos micro serviços poderem ser replicados e terem elasticidade, a arquitetura final possuirá uma alta disponibilidade dos serviços, uma vez que os micro serviços são auto gerenciáveis conforme a demanda de uso.

- **Robustez:** A abordagem de micro serviços também traz benefícios quanto à robustez, uma vez que a tolerância a falhas é beneficiada devido ao uso de *containers* e processos independentes, pois um único micro serviço é completamente isolado de outros micro serviços e só pode ser afetado por eles através das suas interfaces ou através dos recursos dos quais que ele depende, ou seja, mesmo que um micro serviço venha a falhar, outros serviços não serão necessariamente afetados.

Outro ponto interessante a se ter conhecimento é em relação a diferença entre *Service-Oriented Architecture (SOA)* e micro serviços. A principal diferença está em seu tamanho e principalmente em conseguir seguir os princípios definidos por Newman (2015) e Lewis e Fowler (2014), os quais tornam um serviço pequeno, independente, descentralizado, entre outros fatores citados anteriormente. Vale ressaltar que existem pessoas, como por exemplo, Stephen O'Grady (O'GRADY, 2017), principal analista e fundador da empresa RedMonk, e Jeppe Cramon (CRAMON, 2017), especialista em integração de sistemas em larga escala e SOA, que defendem que o tamanho do micro serviço não é o principal fator decisivo na diferença entre SOA e micro serviços, e que a forma de utilizar um micro serviço (como por exemplo, o meio de comunicação entre os serviços) é algo mais importante do que simplesmente projetar seu tamanho.

2.3 Considerações Finais

Neste capítulo foram apresentados os principais conceitos aplicados ao uso de SPL e micro serviços, bem como suas vantagens e desvantagens. Quanto à SPL, o reúso é a principal estratégia adotada em seu processo de desenvolvimento de sistemas que pertencem a uma mesma família de sistemas. Essa abordagem é geralmente utilizada para resolver a questão do gerenciamento de múltiplas versões da linha de produto em aplicações que possuem uma grande variabilidade de funcionalidades. Além disso, o ponto chave para o sucesso de uma linha de produto está relacionada com a gestão da variabilidade do software, que está diretamente relacionada com a evolução e customização da linha, sendo o modelo de *features* uma das abordagens para se realizar essa representação. Entre as principais vantagens dessa abordagem pode-se citar: uma maior lucratividade, uma vez que são criados produtos específicos para cada segmento de mercado; uma maior produtividade, reduzindo custo de desenvolvimento e tamanho de equipe; e também, uma maior qualidade do software, devido ao reúso, que geralmente tende a reduzir de número de defeitos relatados. Por outro lado, a principal desvantagem dessa abordagem está relacionada à implantação do seu uso dentro da empresa, pois como envolve diversos setores da empresa nesse processo, podem ocorrer resistências por algumas partes, além de poder ocorrer uma falta de compromissos da equipe e/ou gerência e também uma interação insuficiente entre as equipes.

O estilo arquitetural de micro serviços é uma abordagem de desenvolvimento onde uma aplicação é composta de um conjunto de pequenos serviços. Essa abordagem é uma forma de conseguir deixar o processo de evolução mais leve, uma vez que é necessário realizar apenas a atualização do micro serviço específico e não de toda aplicação, como ocorre em aplicações monolíticas. Entretanto, ainda não existe uma definição do tamanho ideal de um micro serviço. O que existem são algumas diretrizes criadas por Newman (2015) e Lewis e Fowler (2014) que auxiliam engenheiros de software a projetarem os

micro serviços. Entre esses princípios, os autores recomendam que os serviços devem ser descentralizados, devem ocultar detalhes de implementação internas, devem isolar falhas e que os serviços devem ser modelados em torno de conceitos de negócios, entre outros princípios. Entre as principais vantagens, pode-se citar a escalabilidade e facilidade de implantação do software. Em contrapartida, a sua complexidade de desenvolvimento, suas chamadas remotas, e seu gerenciamento em relação aos múltiplos bancos de dados e transações, são as principais desvantagens dessa abordagem.

Além disso, desenvolver para micro serviços envolve pensar muito bem em: escopo, pensando o que ficar dentro e fora de cada micro serviço; acoplamento, onde não basta separar uma aplicação em partes, pois um micro serviço deve ser usável fora de seu contexto inicial, portanto as interfaces devem ser auto-contidas e auto-descritas, com protocolos independentes; e detalhes de comunicação, como por exemplo, código para publicar as interfaces, traduzir os parâmetros de entrada/saída, o que leva mais tempo e exige mais esforço no desenvolvimento.

Os autores Bubak e Gomaa (2008) ainda mencionam a importância e o ganho competitivo que a união das abordagens de SPL e SOA trazem para as empresas. Entre as principais análises, o fator suporte para reutilização e variabilidade foram que tiveram maior ligação no estudo. Uma vez que SPL foi criada especialmente para planejar e desenvolver produtos que permitem a personalização em tempo de *design*, SOA tem a missão de interligar negócios por meio de serviços pouco conectados conforme suas variabilidades, e isso ocorre uma vez que, SOA emprega alguns dos mesmos padrões usados nos projetos arquitetônicos de uma SPL, como por exemplo, padrões estruturais e de comunicação, o que acaba levando a um desenvolvimento mais ágil e flexível, quando ocorre a aplicação de ambas abordagens juntas.

Por fim, pode-se concluir que para determinados projetos a combinação das abordagens de SPL e micro serviços podem gerar grandes benefícios. Enquanto a linha de produto tende a facilitar a gestão e desenvolvimento de sistemas com muita variabilidade, a integração de micro serviço pode tornar o processo de evolução muito mais fácil, pois a linha é projetada em pequenos serviços, facilitando a manutenção e escalabilidade futura da aplicação. Por outro lado, deve-se tomar cuidado ao projetar software assim, pois o sucesso das evoluções futuras vai depender de como a linha é projetada inicialmente, e para isso, é necessário ter uma visão maior do contexto da linha de produto, para conseguir projetar da melhor forma cada micro serviço.

3 Revisão da Literatura e o Estado da Arte

3.1 Metodologia de Pesquisa

Neste trabalho, foi feita uma revisão da literatura sem seguir nenhum mecanismo sistemático. Não foi feita uma documentação detalhada das strings de busca, pois a pesquisa foi bastante exploratória e envolveu a experimentação com diferentes *queries* nas bases de dados acadêmicas. Além disso, foi pesquisado os trabalhos dos últimos cinco anos (2013 - 2018) nos *Proceedings SPLC*, que é a principal conferência Linha de Produto de Software (SPL). Destaca-se entre as principais bases pesquisadas, a *IEEE Xplore*, *ACM* e *Springer*, pois essas bases, além de possuir uma grande indexação de trabalhos relevantes, possuem áreas com foco na pesquisa desse trabalho. Também, outro critério adotado foi em selecionar artigos com no mínimo quatro páginas, pois tendem a dissertar sobre os objetivos e resultados alcançados de maneira direta e clara, portanto, esses são uma fonte interessante de referência quando se está em busca de realizar um levantamento rápido do estado da arte sobre determinado assunto. Por fim, o processo de busca foi dividido em duas etapas principais, que são:

- **SPL e Micro Serviços:** A primeira etapa foi executada com o intuito de encontrar trabalhos que abordassem micro serviços e SPL, o qual se aproximam da proposta deste trabalho. Porém, não foram encontrados muitos artigos com esse foco, apenas dois.

- **Outros Trabalhos Relacionados:** A segunda etapa foi focada em encontrar artigos que abordassem o uso de serviços na evolução de uma linha de produto. Nessa busca, foram encontrados diversos trabalhos, e com isso, optou-se por detalhar bem apenas os principais trabalhos relacionados (que abordassem serviços e SPL na evolução de uma linha de produto) a presente pesquisa.

3.2 Trabalhos Relacionados

Não foram encontrados na literatura muitos artigos que combinam as técnicas de SPL e micro serviços. Tal ausência também foi notada por Tizzei et al. (2017). Com isso, foram encontrados apenas dois artigos, sendo um deles diretamente relacionado com este trabalho de mestrado, que é o artigo de Tizzei et al. (2017). Na subseção 3.2.1 apresenta-se os dois artigos que utilizaram ambas as técnicas, e na subseção 3.2.2 aborda-se artigos relacionados a SPL e serviços, evolução e demais trabalhos relacionados a este.

3.2.1 Técnicas de SPL e Micro Serviços

No trabalho de Tizzei et al. (2017), os autores investigaram o uso de técnicas de SPL e arquitetura de micro serviços para dar suporte à reutilização de software, aplicando essas técnicas na reengenharia do sistema legado WISE, da IBM. Tratava-se de um sistema multi-inquilino[1] (em inglês multi-tenancy) que começou a apresentar problemas de manutenção e desempenho. Com isso, o principal objetivo foi reduzir o esforço de manutenção e melhorar o suporte à escalabilidade do sistema.

Segundo os autores, o uso de micro serviços por si só não tem como foco principal gerenciar a variabilidade, o que era essencial para proporcionar a reutilização desejada, uma vez que era um sistema multi-inquilino. Além disso, os autores consideram como importante a configuração da linha de produtos. Nesse sentido, o trabalho utiliza a modelagem de *features* (KANG et al., 1998) como forma de alcançar tais objetivos.

Portanto, a implementação do novo sistema foi guiada através de um processo de reengenharia orientado a *features*, sendo primeiramente criados os modelos mais abstratos (arquitetura legada, modelo de *features*) a partir de artefatos legados. Após essa primeira etapa, foi executada uma análise de domínio, com o intuito de definir um novo modelo de *features*. No final, com base nesse novo modelo e na arquitetura herdada, uma nova arquitetura de software foi projetada e implementada.

Para dividir os micro serviços nessa nova arquitetura, os autores resolveram manter os elementos arquitetônicos do software legado (*front-end* WISE-UI e *back-end* WISE-Parser) como serviços únicos cada um, pois não estava claro na época se a divisão em serviços menores iria aumentar a modularidade ou se iria ter um impacto maior na perda de desempenho do sistema. Tal raciocínio está de acordo com as diretrizes de Lewis e Fowler (2014): em situações que podem ocasionar perda de desempenho é recomendado manter uma granularidade grossa do serviço. Além disso, utilizaram serviços de terceiros em nuvem para persistência do dados, o que auxilia para estar em conformidade com a diretriz que fala que o serviço precisa ser descentralizado.

Segundo os autores, as técnicas de SPL auxiliaram na reutilização e na manutenção do software, enquanto que a arquitetura de micro serviços auxiliou no suporte à escalabilidade e na evolução independente dos serviços. A avaliação das técnicas ocorreu num período de 384 dias e no final obtiveram resultados positivos, atingindo o objetivo inicial do trabalho, ou seja, a manutenção e escalabilidade foi observada como mais fácil. Além disso, conseguiram reutilizar em média 62% de linhas de código por inquilino. Porém, tiveram alguns *trade-offs* no projeto da nova arquitetura devido à mudança da abordagem centralizada para descentralizada, pois apesar da arquitetura de micro serviços favorecer uma abordagem descentralizada para promover a implantação de serviços independentes,

[1] Multi-tenancy ou multi-inquilino é a capacidade da aplicação suportar a execução de diversos usuários sem comprometer a performance, segurança entre os envolvidos.

cada serviço precisou de uma representação própria das *features* para cada inquilino, uma vez que não há mapeamento global de inquilinos para as *features*. Por fim, deixaram como trabalhos futuros investigar a abordagem proposta a outros micro serviços e analisá-los de outras perspectivas, além da reutilização de software, como também na evolução de software, na confiabilidade e no desempenho.

Como demonstrado no artigo, a utilização de SPL e micro serviços pode trazer grandes benefícios para sistemas que desejam conseguir uma maior escalabilidade e principalmente na melhora do esforço em relação à manutenção do software. Porém, devido ao uso de micro serviços ser uma abordagem recente, ainda não existe um consenso ou padronização com relação ao tamanho ideal de um serviço, ficando a cargo da equipe e da experiência de cada membro analisar e decidir a granularidade de cada serviço.

Apesar de elucidar vários pontos interessantes para esta pesquisa, o artigo aborda de forma resumida o processo de como foram projetados os serviços, portanto este ponto precisa de mais investigação. Além disso, o principal foco da avaliação foi em tarefas de manutenção, e não evolução, como se pretendeu investigar nesta pesquisa.

Naily et al. (2017) propuseram um *framework*, denominado *ABS Microservices Framework*, para melhorar a flexibilidade da variabilidade da família do produto, utilizando as abordagens de Engenharia de Linha de Produto de Software (SPLE) e micro serviços. Esse *framework* foi projetado para trabalhar de forma semelhante a outros *frameworks*, como por exemplo o Spring Boot[2], porém com uma melhor flexibilidade para lidar com as variabilidades dos requisitos. Além disso, o *framework* foi desenvolvido utilizando o Abstract Behavioral Specification (ABS), que é uma maneira formal para modelar software com um alto nível de variabilidade. Neste trabalho, os autores utilizaram o processo SPLE para criar um produto baseado em micro serviços utilizando ABS. Este processo é dividido em engenharia do domínio e engenharia da aplicação. A engenharia do domínio é responsável por analisar a variabilidade dos requisitos e criar artefatos reutilizáveis, e é dividida na análise do domínio, que analisa o escopo e todas as possibilidades do domínio e tem como saída um conjunto de *features*, e na implementação do domínio, que com base na análise do domínio, implementam os artefatos reutilizáveis, gerando o código fonte. Esses artefatos reutilizáveis são implementados com um módulo *core* e *deltas*. O módulo *core* é a base do produto e contém pontos em comum do produto, e os *deltas* são um conjunto de módulos *delta* que implementam a variabilidade do produto, representando uma ou mais *features*. A engenharia da aplicação é responsável por reutilizar os artefatos e se divide em análise de requisitos e derivação do produto. Na análise de requisitos são selecionadas as *features* com base nas necessidades de cada cliente, enquanto a derivação do produto é o processo para compor o produto conforme as *features* selecionadas na

[2] Spring Boot - é um projeto da Spring que visa facilitar o processo de configuração e publicação das aplicações (https://projects.spring.io/spring-boot/)

etapa anterior, gerando assim o produto baseado em micro serviços de acordo com a sua configuração.

Com isso, para representar os micro serviços, os autores adicionaram no módulo *core* o módulo micro serviços, ou seja, o módulo de micro serviços é o módulo que contém a implementação dos micro serviços, permitindo assim, implementar um ou mais micro serviços. Esse módulo é composto por cinco camadas, que são: a camada de recursos, responsável por controlar a solicitação recebida e as mensagens para os objetos que representam o domínio; a camada de serviços, que contém a lógica de implementação; a camada de domínio, que representa o modelo ou entidade relacionado com o micro serviço; a camada de repositório, que provê as operações de acesso aos dados; e a camada de persistência, que mapeia os dados do domínio. Por fim, o *framework* possui um *middleware* que atua como uma interface entre os micro serviços e os sistemas externos.

Como trabalho futuro, está a implementação dos métodos Hypertext Transfer Protocol (HTTP) *PUT* e *DELETE*, a representação de outros formato dos dados, como XML, a integração com ferramentas de teste, o suporte a vários bancos de dados, além de finalizar o projeto do *framework*, a fim de melhorar a comunicação das *features* e dos micro serviços.

Apesar dos autores mencionarem que o *framework* proposto tem uma melhor flexibilidade diante de outros *frameworks* que já estão em produção no mercado e que o projeto está em fase de evolução, ainda é necessário uma melhor avaliação em um cenário real da indústria para conseguir um feedback válido quanto a essa real flexibilidade, porém, essa avaliação e aceitação pela indústria é dificultada pois a utilização de formalismo matemático acaba elevando o custo do projeto e necessitando especialistas no assunto, ou seja, que os membros da equipe tenham um conhecimento em métodos formais. Além disso, a utilização de micro serviços foge dos princípios mencionados na seção 2.2, pois os serviços não estão de fato isolados e existe um forte acoplamento dos serviços.

3.2.2 Outros Trabalhos Relacionados

No trabalho de Capilla e Topaloglu (2005), os autores tentaram melhorar o desenvolvimento e a evolução de sistemas orientados a serviços (SOA), propondo uma modificação do processo tradicional de SPLE, adicionando informações de variabilidade específicas, a fim de suportar melhor a composição dos serviços.

Nessa modificação do processo, na fase de engenharia de domínio, foi incluída na criação da arquitetura de software informações específicas (pontos de variabilidades) para suportar a composição do serviço e tanto os serviços simples como os serviços compostos podem ser projetados nessa fase. Na fase de engenharia da aplicação, são utilizados os serviços criados na fase anterior, a fim de desenvolver a aplicação orientada para serviços.

Para as aplicações que utilizam serviços únicos, é feito um link direto da arquitetura para o serviço específico, e no caso de possuírem serviços compostos, podem ser construídos na fase de engenharia de domínio ou através de um processo de composição dos serviços, antes do produto final ser entregue. O objetivo do processo de composição do *web service*[3] é personalizar serviços compostos através de pontos de variação específicos, selecionar os serviços certos a serem compostos e realizar uma avaliação preliminar antes do seu uso.

Além disso, foi utilizada uma arquitetura de linha de produto leve, pois ela é aplicada a abordagens ágeis e para negócios que possuem um tempo de entrega pequeno para ir ao mercado. Entende-se por linha de produto leve uma abordagem incremental em que o número de práticas específicas de SPL utilizado é pequeno (CAPILLA; DUEÑAS, 2005).

Um dos pontos-chave para o sucesso da reutilização bem-sucedida é a identificação correta das informações de variabilidade, as quais são usadas para descobrir os aspectos comuns e variáveis em todo o sistema, e que pode gerar um custo alto durante o desenvolvimento, caso não seja projetado corretamente no início do projeto. Com isso, os autores propuseram vários pontos de variação para modelagem de serviços web, que são: modelo de orquestração, que define a sequência de atividades que ocorrem durante a execução do serviço; o modelo de dados, que define os dados trocados entre os serviços; a seleção do serviço, que especifica a vinculação entre os serviços ao criar um serviço composto; o gerenciamento de exceções, que gerencia as exceções durante a invocação do serviço; e os fatores de qualidade também podem ser definidos, como por exemplo, custo, desempenho e disponibilidade. Toda essa informação deve ser definida na arquitetura do software. Além disso, os diagramas de classes UML são usados para descrever a relação entre os elementos da arquitetura e os pontos de variação, enquanto que na visão dinâmica das possibilidades são utilizados diagramas de atividade UML.

Por fim, as duas principais contribuições do trabalho foram: modificar a abordagem tradicional de SPL e incluir um processo específico para a composição de serviços, e em segundo, definir pontos de variabilidade específicos na arquitetura para especificar diferentes alternativas para compor esses serviços. Como resultado disso, com a utilização de uma linha de produto leve e através da variabilidade específica na descrição do serviço, conseguiram melhorar a evolução dos sistemas orientados a serviços. Porém, além de não apresentar uma métrica que comprove uma real evolução do software com a abordagem proposta, os autores ainda mencionam como trabalhos futuros uma validação dos pontos de variabilidade definido na arquitetura pelo processo proposto, sendo também necessário testar esses serviços em diferentes cenários.

Horcas, Pinto e Fuentes (2016) propuseram uma abordagem usando a Arquitetura de Linha de Produto (PLA) para gerenciar a variabilidade e a evolução de aplicações

[3] Web Service é uma solução utilizada para realizar a comunicação externa dos serviços entre aplicações.

multi-tenancy em nuvem, através de modelos de variabilidades baseados na Linguagem Comum de Variabilidade (CVL). Além disso, os autores apresentaram três algoritmos para evoluir automaticamente uma configuração prévia dos inquilinos, calcular as mudanças necessárias na arquitetura e propagar a evolução para a arquitetura *multi-tenancy*, utilizando transformações Model-To-Model (M2M).

O modelo CVL permite implementar toda a variabilidade do sistema, separando em uma árvore as *features* que são obrigatórias e as opcionais. As *features* opcionais são os pontos de variações de cada inquilino. Cada ponto de variação é mapeado com uma característica na árvore e referenciado a um ou mais elementos da arquitetura. Também, para dar suporte às diferentes configurações do sistema de cada inquilino, a abordagem definiu a funcionalidade específica de cada inquilino em *features* clonáveis, as quais possuem uma cardinalidade que indica se a *feature* pode ser ou não instanciada, e consequentemente, todas as *features* da sub-árvore poderão ser configuradas conforme a necessidade de cada inquilino.

O processo de evolução consiste basicamente em modificar o modelo de variabilidade com as novas *features* (seja adicionando, removendo ou atualizando elas), e após isso, realizar a propagação dessas mudanças na configuração de todos os inquilinos existentes. Para automatizar esse processo de evolução, a etapa de propagação é dividida em dois passos, onde serão executados os três algoritmos propostos pelos autores. A primeira parte é a execução de dois algoritmos: o *Evolve Configuration Algorithm*, que é responsável por modificar a configuração atual de todos os inquilinos e produzir uma nova configuração evoluída, e o *Difference Configuration Algorithm*, que é responsável por calcular as diferenças entre a configuração evoluída e a anteriormente implementada. Por último, o passo mais difícil é a execução do algoritmo *Create Weaving Model Algorithm*, que consiste em propagar automaticamente as mudanças de evolução calculadas pelos algoritmos anteriores para a arquitetura de software implantada.

Portanto, a principal contribuição desse trabalho foi a criação do terceiro algoritmo apresentado anteriormente. Além disso, o trabalho está em evolução, e tem como trabalhos futuros realizar automaticamente a implantação da arquitetura evoluída na nuvem, realizar a avaliação quanto aos esforços de manutenção, escalabilidade, entre outros. Entretanto, apesar do trabalho dos autores também investigar a evolução de uma linha de produto *multi-tenancy*, essa é uma abordagem que depende fortemente da modelagem da variabilidade em features e conseguir fazer o mapeamento M2M funcionar, ou seja, ter a facilidade na manutenção é algo trabalhoso e que exige um nível de formalidade que nem sempre é possível conseguir em tempo hábil ou pela equipe disponível.

No artigo de Ajila e Kaba (2004), os autores identificaram um mecanismo de gerenciamento de mudanças para apoiar a evolução da SPL e desenvolveram um modelo de evolução baseado na estrutura de relacionamentos de dependências dos vários artefatos

da linha de produto. Os mecanismos definidos possuem quatro estratégias em comum, que são: a identificação de mudanças, que se baseia no histórico da mudança para saber o motivo que levou à alteração; o impacto de mudanças, onde a partir dos relacionamentos das dependências do artefato é calculado o impacto em todos os outros artefatos da linha; a propagação de alterações, que é a propagação da mudança em toda base da linha, a partir de um prévio conhecimento de como será realizada a mudança e seu responsável; e por último a validação de alterações, onde é realizada uma validação para garantir a consistência da alteração. Além disso, essas estratégias são executadas em três processos de gerenciamento de mudanças dos artefatos, que são: o processo de mudança da arquitetura da linha de produto, que tem como principal objetivo projetar uma arquitetura que engloba todos os produtos e seus recursos compartilhados entre eles; o processo de mudança dos componentes, que são os componentes identificados para serem reutilizados na arquitetura da linha de produtos; e por último, o processo de mudanças do produto, que inclui todos os produtos desenvolvidos com base na arquitetura e nos componentes.

Ainda é apresentado um modelo conceitual para o processo de evolução de SPL. O principal conceito aplicado nesse modelo é o espaço de referência da linha de produto (*Space - PLRS*), que é um metamodelo que apresenta o contexto do processo de desenvolvimento, além de definir os espaços de trabalho para arquitetura de produtos, a linha de produtos, os componentes reutilizáveis, e ainda especificar os tipos de entidade dos artefatos e suas relações, entre outras informações.

Por fim, é apresentado um método para dar suporte ao modelo de evolução proposto. Esse método possui diversas operações definidas, como *Add, Delete, Update, Create e Evaluate*, entre outros, e tem como principal objetivo facilitar a identificação das entidades de evolução e o gerenciamento do processo de evolução. O modelo de evolução, que faz uso desse método e dos mecanismos de rastreabilidade, consiste em duas bases: a base de ativos principais e a base da lista de dependências da estrutura. Através de uma ferramenta para gerenciar as mudanças, o usuário pode interagir com a base a fim de verificar o conteúdo da lista gerada e poder modificá-la. Além disso, essa ferramenta para gerenciar mudanças é controlada pelo processo de mudanças apresentado anteriormente, e tem como saída a SPL evoluída. Dessa forma, as principais contribuições do artigo são o mecanismo de rastreabilidade que auxilia a evolução da SPL e o modelo de evolução, que tem como principal objetivo aumentar a produtividade do desenvolvedor da linha de produtos e do gerente de processo de desenvolvimento, através das informações sobre a evolução do objeto contida no modelo. Entretanto, num cenário orientado a micro serviços, espera-se que a informação de rastreabilidade seja menos importante para a manutenção ou evolução de uma SPL, uma vez que os micro serviços tendem a gerar uma estrutura menos acoplada e mais coesa. Ainda assim, a abordagem proposta pelos autores pode ser útil, pois deverá haver algum tipo de mapeamento entre as *features* e os micro serviços.

No trabalho de Teixeira, Borba e Gheyi (2015), os autores definiram as bases para uma evolução segura e modular de populações de produtos e múltiplas SPL, através da teoria do refinamento da linha de produtos e das propriedades de composição. Uma população de produtos é um conjunto de famílias de produtos que compartilham recursos e abrangem domínios com funcionalidades sobrepostas para suportar o desenvolvimento de características separadas. E as múltiplas linhas de produtos combinam e compõem outras linhas de produtos e são desenvolvidas de forma independente, mas dependem uma da outra para gerar um produto.

A teoria de refinamento formaliza a evolução por meio da orientação de quando o comportamento deve ser mantido após as mudanças ou quando desejam que as alterações não propaguem nos produtos já existentes. Essa teoria define uma linha de produto como três elementos principais, que são: o modelo de variabilidade; um mapeamento dos ativos bases; e um mapeamento dos recursos para saber as configurações dos produtos. Com isso, são utilizados modelos de variabilidade e de configuração, e é modular através da composição da relação de mudanças de cada elemento. Os refinamentos são transformações seguras, no sentido de ser possível alterar uma linha de produto sem impactar nos usuários existentes, pois o comportamento dos produtos existentes é preservado uma vez que o refinamento da linha de produto é composicional, ou seja, permite modificar a linha de produto de forma independente sob certas condições estabelecidas.

Entretanto, é necessário uma formalização bem definida dos modelos e dos componentes modulares para conseguir controlar uma evolução segura das múltiplas SPL. Por fim, ainda encontra-se em aberto a investigação dos problemas de modularidade para uma evolução segura das populações de produtos e das múltiplas linhas de produtos, em relação aos padrões de evolução e da derivação dos modelos de refinamento.

No trabalho de Schubanz et al. (2013), que é uma evolução do trabalho de Schubanz et al. (2012), o qual faz parte do mesmo grupo de pesquisa, os autores apresentam uma abordagem denominada *EvoPL* para planejar e monitorar a evolução de uma SPL, através de técnicas orientadas por modelos, a fim de criar um modelo conceitual e dar suporte para ferramentas.

A proposta dos autores está ligada com a evolução a longo prazo da linha de produto e tem seu escopo vinculado em duas dimensões: no tempo, o qual apoia o planejamento contínuo por longos períodos de tempo e muitas versões (*releases*); e no espaço, pois dá suporte às decisões de alto nível até os artefatos de implementação.

Essa abordagem permite fornecer suporte no tempo, planejamento contínuo e em diversos lançamentos da SPL. Além disso, o suporte a ferramentas permite rastrear informações de planejamento até a implementação, resolução de *bugs* e *commits*. Entretanto, como os autores mencionam que essa abordagem ainda possui algumas limitações (como por exemplo, nem todas as partes do modelo podem ser editadas) ainda é necessário uma

evolução da mesma, principalmente em relação ao suporte de planejamento e fornecer uma avaliação mais formal.

3.3 Considerações Finais

Diante do levantamento apresentado neste capítulo, pode ser afirmado que as abordagens de micro serviços e SPL ainda carecem de investigação por trabalhos acadêmicos, além de ser um tema bastante relevante nos dias atuais. Na maioria dos trabalhos encontrados os autores abordam a utilização de SPL e/ou arquitetura de serviços tradicional, o qual não tem a preocupação com tamanho dos serviços. Em relação ao tamanho dos micro serviços, ainda não existe um consenso que estabeleça qual o tamanho ideal de cada um. O que existe e auxilia os engenheiros de software a projetarem esses micro serviços são os princípios propostos por Newman (2015), Lewis e Fowler (2014), e que mencionam que um serviço deve ser mínimo possível em relação a uma regra de negócio específica e que seja independente e tolerante a falhas, entre outros fatores discutidos no Capítulo 2.

Um dos indicativos de que SPL e micro serviços podem ser uma boa combinação para melhorar as tarefas de evolução de uma linha de produto é o fato de que os autores Tizzei et al. (2017) tiveram benefícios em relação à manutenção de uma linha de produto. As técnicas de SPL auxiliaram na reutilização e na manutenção do software, enquanto a arquitetura de micro serviços auxiliou no suporte à escalabilidade e na evolução independente dos serviços. Além disso, os autores deixaram como trabalhos futuros justamente a avaliação da evolução de uma SPL.

Por fim, evidências empíricas são sempre buscadas pelos pesquisadores e este trabalho vai exatamente nessa direção, propondo avaliar a aplicação das abordagens de micro serviços e SPL na evolução de uma linha de produto, em um cenário real da indústria de software.

4 O Impacto do Uso de Micro Serviços na Evolução de uma Linha de Produto de Software

Conforme mencionado no Capítulo 1, esta pesquisa de mestrado teve como principal objetivo avaliar a evolução (tanto na adição de novas *features*, como na manutenção de *features* existentes) de uma Linha de Produto de Software (SPL), através da utilização de micro serviços, para no final conseguir avaliar se o impacto de micro serviços realmente melhorou a tarefa de evolução. Sendo assim, para que fosse possível medir o impacto de micro serviços em uma SPL, foi realizado um estudo de caso comparativo entre dois cenários diferentes. Esses dois cenários foram projetados em parceria com uma empresa[1] sediada no Rio Grande do Sul e que possui em seu portfólio de produtos um Sistema de Gestão Empresarial (ERP) implementado em uma linha de produto, que é gerenciada de maneira *ad-hoc*, ou seja, não utiliza recursos para auxiliar nesse processo, como por exemplo, o uso da modelagem de *features*. A escolha da Empresa X se deu devido primeiramente pelo fato dela possuir um potencial produto de SPL para aplicação de estudo deste trabalho e de sofrerem com os principais problemas levantados no Capítulo 1, como por exemplo, a dificuldade em realizar o *deploy* a cada evolução das aplicações de todos os seus clientes. Além disso, a escolha também foi por conveniência, pois o autor deste trabalho é colaborador da empresa. Por último, a empresa mostrou um grande interesse em avaliar o uso de micro serviços juntamente com o uso correto de SPL em seu produto, fornecendo assim recursos do seu software, equipe de desenvolvimento e um ambiente para aplicação da avaliação.

Projetar uma SPL é algo trabalhoso principalmente no início do desenvolvimento, pois é necessário ao projetista ter uma visão mais ampla do produto, para assim conseguir criar uma arquitetura adequada e conseguir separar as *features* para que no futuro seja possível uma evolução segura, rápida e fácil. Segundo Linden, Schmid e Rommes (2007), a principal tarefa que ditará o sucesso ou o fracasso de uma abordagem de engenharia de SPL é o gerenciamento de variabilidade, o qual representa todo o escopo, similaridades e as variabilidades da linha, juntamente com ferramentas que automatizem esse processo.

Outro problema inerente a este trabalho é em relação a abordagem de múltiplos inquilinos e múltiplos micro serviços. Quando se trabalha com múltiplos inquilinos, uma das abordagens (a qual é utilizada hoje pela Empresa X) é a geração de vários arquivos de *deploy* (nesse caso, como o produto é desenvolvido em java web, o arquivo é o Web appli-

[1] No restante deste trabalho, será utilizado o termo Empresa X em referência à empresa parceira.

cation ARchive (WAR)), um para cada cliente, apenas com as funcionalidades escolhidas pelo cliente. Essas funcionalidades podem ser liberadas através de diretivas de compilação com o uso dos comandos *if-else*, com a compilação do WAR com as *features* específicas, ou até mesmo com a união de ambas as técnicas. Porém, essa estratégia de geração de vários arquivos de *deploy* apesar do benefício de permitir uma independência maior entre os clientes, pode tornar o processo de evolução trabalhoso e difícil de ser realizado, pois qualquer alteração de uma regra de negócio para correção de erros, por exemplo, requer uma atualização de toda a aplicação e em todos os clientes que a utilizam, levando um esforço de tempo considerável e um custo alto para a empresa. Por outro lado, a partir do momento que a ideia de micro serviços é implantada, todas as regras de negócio deverão ser divididas em vários micro serviços, enquanto que no arquivo de *deploy* deverá ficar apenas a camada de visualização da aplicação, o que torna o processo de evolução mais simples e rápido, pois com a evolução de qualquer funcionalidade, é necessário apenas atualizar o micro serviço específico uma única vez, e todos os clientes estarão com a aplicação atualizada.

Para entender melhor a solução da Empresa X, foi criado um modelo de *features* com uma parte da solução da sua família de produtos, conforme apresentado na Figura 1. A linha de produtos possui três *features* obrigatórias, que são: Segurança, que é responsável por toda a segurança do sistema e gestão dos multi-usuários; Licença, que faz o controle das multi-empresas do sistema e também é responsável por gerenciar a licença contratada pelo plano do cliente, ou seja, faz o controle de quais *features* estão ou não liberadas para o usuário; e a última *feature* é a Cadastro, a qual possui os principais cadastros bases pertinentes a qualquer outra *feature*. Nessa *feature*, existe uma regra (*require*) que define que, caso no cadastro de produto seja liberado o preenchimento de informações da parte tributária do produto, é obrigatório a seleção da *feature* Tributário. O mesmo vale quando a *feature* Documento Fiscal Eletrônico (DFe) é selecionada: a *feature* Tributário também precisará ser selecionada. Além das *features* obrigatórias, existem mais três *features* opcionais, que são: Financeiro, que é responsável por todas as funcionalidades de operações financeiras; Tributário, que é responsável por toda configuração tributária para emissão de qualquer documento fiscal; e a *feature* DFe, que é responsável por todos os documentos fiscais eletrônicos, como por exemplo, NFe (Nota Fiscal Eletrônica), NFCe (Nota Fiscal do Consumidor Eletrônica) e CTe (Conhecimento de Transporte Eletrônico).

Vale destacar que, a partir deste modelo de *features*, já se tem uma boa ideia das configurações dos produtos/clientes, e isso pode facilitar a manutenção/evolução. Entretanto, neste trabalho o objetivo foi ir além, explorando a aplicação de micro serviços para medir o real impacto dessa abordagem na tarefa de evolução de uma SPL.

Com o entendimento da solução da Empresa X serão apresentados os dois cenários que serviram como base de avaliação do uso de micro serviços.

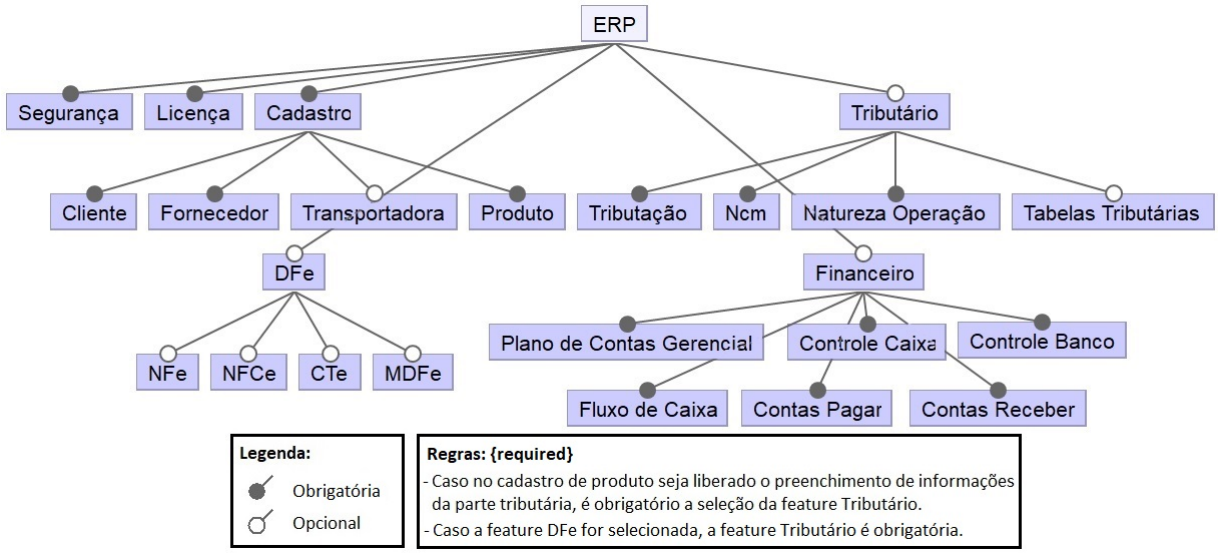

Figura 1 – Sistema Atual - Modelo de *Feature*

No cenário 1, que é o cenário atual da Empresa X e que está em produção em seus clientes, a solução foi desenvolvida com a tecnologia Java Web, com JavaServer Faces (JSF) e uma arquitetura Model–View–Controller (MVC). Para a camada de visualização, foram utilizados os *frameworks* PrimeFaces e Bootstrap, enquanto que na camada de persistência foi utilizada a especificação Java Persistence API (JPA) com o *framework* Hibernate e banco de dados PostgreSql. Conforme apresentado na Figura 2, cada cliente tem sua própria aplicação, ou seja, é gerado um arquivo WAR para cada um, e na medida que ocorre a evolução da linha, é necessário realizar a atualização em cada cliente, uma vez que o WAR possui as funcionalidades específicas de cada cliente. Além disso, vale relembrar que a Empresa X não utiliza recursos para automatizar e gerenciar a linha de produto.

Na Figura 3 é apresentado o cenário 2, que é o cenário onde o autor deste trabalho implementou a aplicação utilizando a abordagem de SPL e micro serviços. Para fins de um melhor reaproveitamento futuro por parte da Empresa X, foram utilizadas as mesmas tecnologias e *frameworks* apresentados no cenário 1, ou seja, a aplicação web foi desenvolvida com JSF e os *frameworks* PrimeFaces e Bootstrap, e no back-end (micro serviços), foi utilizado o *frameworks* Spring MVC (com o plugin Spring Boot) e Hibernate, com banco de dados PostgreSql. Porém, foi projetado um único WAR para seus clientes, o qual contém apenas o conteúdo da camada de visualização e as chamadas aos micro serviços que implementam as regras de negócios da aplicação. Com isso, conforme é feita a evolução da aplicação, é necessário apenas atualizar o micro serviço específico, que o consumo pela aplicação web continuará funcionando normalmente. Por fim, conforme a literatura, o uso de micro serviços ainda é um ponto em aberto, já que não se tem bem claro qual é o tamanho/escopo ideal de um micro serviço. Prova disso, é que no

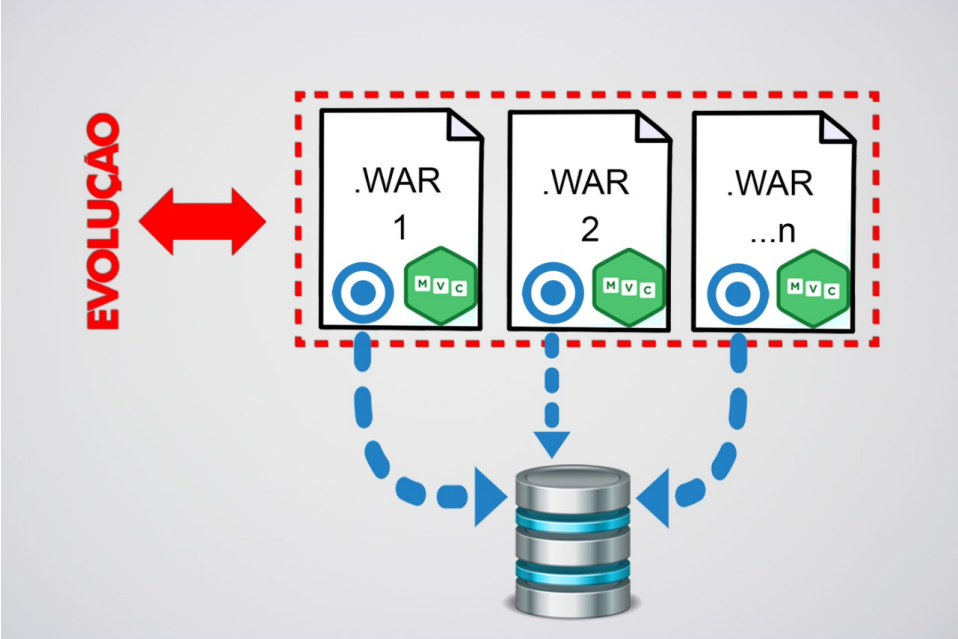

Figura 2 – Cenário 1

trabalho de Tizzei et al. (2017), que é o trabalho que tem a proposta mais próxima da aqui apresentada, também se depararam com esse problema, e ao projetar seus micro serviços tomaram como base a experiência de seus engenheiros de software. Portanto, os micro serviços foram projetados de acordo com as diretrizes de Newman (2015), Lewis e Fowler (2014), apresentado no Capítulo 2.

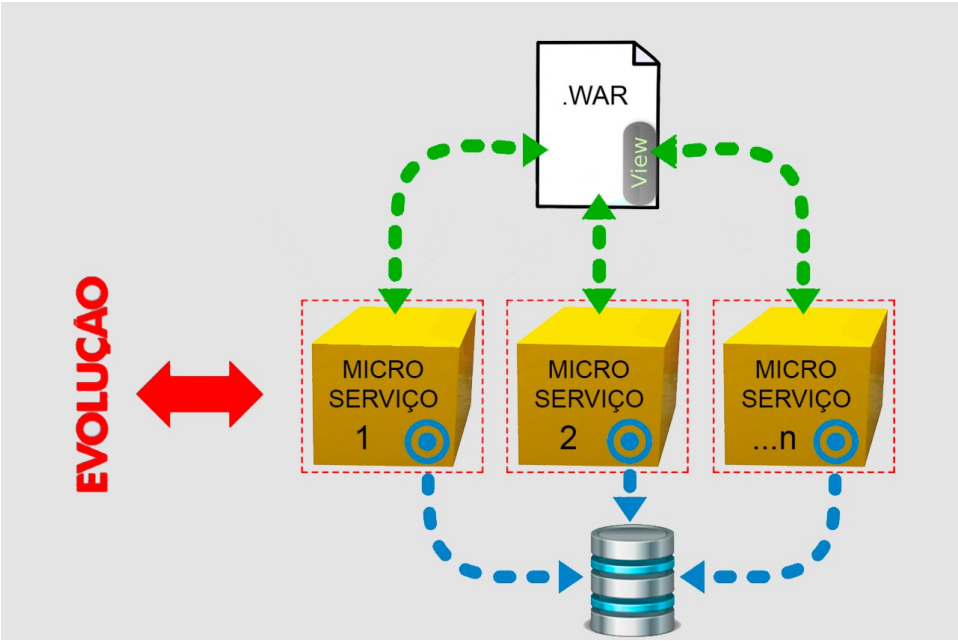

Figura 3 – Cenário 2

4.1 Migração para Arquitetura de Micro Serviços

Considerando que era necessário que os dois cenários estivessem desenvolvidos para que o experimento pudesse ser executado e, assim, conseguir realizar a avaliação proposta deste trabalho, o autor dessa dissertação teve que realizar a implementação da estrutura inicial do cenário 2, com base no cenário 1 da Empresa X. Como ainda não existe um consenso na literatura sobre a granularidade de um micro serviço, cada micro serviço foi projetado com base no conhecimento técnico em cima do produto de software e seguindo as diretrizes criadas por Newman (2015), Lewis e Fowler (2014). Além disso, vale destacar que quando o foco é na migração de uma arquitetura sem micro serviço para uma com micro serviço o problema pode ser maior, ou seja, a refatoração de um sistema já existente sem nenhum suporte a micro serviços é mais difícil do que a criação de um software do zero com micro serviço, pois demanda um entendimento do código existente e uma visão geral de como está a arquitetura da linha de produto, para assim, conseguir projetar os micro serviços.

Exemplificando o problema, é possível pensar que funções do tipo CRUD (*Create*, *Read*, *Update* e *Delete*) são simplesmente separadas em micro serviços. No entanto, temos conhecimento que grande parte de sistemas focados na manipulação de informações tem regras de negócios bastante complexas. Logo, o primeiro desafio da migração de um software sem micro serviço para um com micro serviço é a decisão da granularidade e o segundo desafio é a definição de quais regras de negócios ou funcionalidades relacionados com CRUD serão micro serviços.

Seguindo as diretrizes de Newman (2015), Lewis e Fowler (2014), pelas quais os micro serviços devem ser modelados em torno de conceitos de negócios, devem ser descentralizados, devem ocultar detalhes de implantação internas e a implantação ser independente, optou-se pela decisão da granularidade do micro serviço ser por operações de manipulação de registro e também em criar outros micro serviços conforme as pequenas regras de cada contexto.

Analisando o CRUD da entidade Pessoa, por exemplo, é possível salvar, excluir, alterar e pesquisar registros de pessoas. Cada operação de manipulação de registro virou um micro serviço. Além disso, podemos ter regras de negócio específicas. Por exemplo, uma pessoa comumente tem o CPF como um atributo. O CPF necessita ser validado para garantir a integridade do cadastro da pessoa. Essa funcionalidade comumente seria implementada no próprio modelo da entidade Pessoa ou adicionada em uma camada EJB (Enterprise Java Beans), por exemplo. Porém, quando estamos pensando em uma arquitetura de micro serviço, a responsabilidade de validar o CPF seria repassado para um micro serviço, ficando totalmente separada da aplicação com a interface gráfica. Com isso, diferentes dispositivos independentes de linguagem de programação foram criados, e poderão utilizar o serviço de validação de CPF.

Imagine agora a implementação do cadastro de pessoa nos dois cenários descritos na seção anterior. No primeiro cenário, temos uma arquitetura MVC baseada em JSF e EJB. Logo, para criar um cadastro de pessoa seriam necessários os seguintes arquivos:

- Arquivo de interface gráfica Pessoa (Projeto Cliente e Servidor).
- Classe da entidade pessoa (Projeto Cliente e Servidor).
- Classe EJB contendo os serviços de acesso ao banco e manipulação da entidade pessoa (Projeto Projeto Cliente e Servidor).

No outro lado, teremos a arquitetura de micro serviços para manipular o cadastro de pessoa. Logo, para criar um cadastro de pessoa com base na arquitetura de micro serviços seriam necessários os seguintes arquivos:

- Arquivo de interface gráfica pessoa (Projeto Cliente).
- Classe da entidade pessoa (Projeto Cliente e MS).
- Classe denominada *"filter"*, contendo os filtros do serviço (Projeto MS).
- Classe denominada *"repository"*, contendo as operações de banco (Projeto MS).
- Classe denominada *"service"*, contendo as regras de negócio da entidade (Projeto MS).
- Classe denominada *"resource"*, que tem como objetivo especificar os micro serviços da entidade (Projeto MS).

O que podemos notar é que o número de classes para migrar uma entidade para outra é muito maior quando trabalhamos com micro serviços. Vale ressaltar que essa afirmação é baseada no uso do Spring Boot, e não foi avaliado se o mesmo ocorre com o uso de outra tecnologia. Todavia, é necessário notar que o projeto do micro serviço é independente. Além disso, o *framework* possibilita a separação das responsabilidades da manipulação da entidade em filtros, operações no banco, regras de negócios e o serviço. Outro ponto positivo é que no cenário 1 existe um arquivo WAR para cada cliente, então toda modificação feita nas regras de negócio da classe Pessoa deve ser atualizadas em todos os arquivos.

Os pontos negativos da arquitetura de micro serviços ficam por conta de não guardarem nenhum estado, ou seja, cada requisição terá um estado que será apagado quando a requisição terminar. Além disso, também será necessário em cada requisição enviar juntamente com os dados da entidade, qual o nome do banco de dados em que deverá ser feita a modificação. Visto que ao utilizar um único WAR para todos os clientes, no momento

do consumo do micro serviço, deve ser especificado em qual banco as modificações irão acontecer.

Além disso, o processo de migração levou em torno de sete meses para ser finalizado. Estima-se que o autor deste trabalho realizou entre 150 a 200 horas de trabalho, para deixar os dois cenários no mesmo nível de implementação, além de realizar estudos e aprendizados das tecnologias envolvidas nesse mesmo tempo. Durante este trabalho, ao projetar o cenário dois em conjunto com a Empresa X, optou-se em agrupar os micro serviços em projetos maiores, para assim, facilitar a gestão da variabilidade da linha de produto e também a gestão de todos os micro serviços, ou seja, os micro serviços referentes a *feature* Tributária agrupavam todos os micro serviços com as demais *features* pertinentes a esse módulo, o mesmo para a *feature* do Financeiro, Cadastro e demais *features* dos outros módulos do sistema ERP. Essa decisão de projeto pode ir um pouco na contra mão do princípio da Implantação Independente, proposta por Newman (2015), Lewis e Fowler (2014), porém, criar toda uma estrutura para cada micro serviço, na prática, tornaria a gestão muito difícil. Além disso, o foco desta pesquisa está nos aspectos de desenvolvimento, e não da operação. Neste sentido, o fato de a implantação ser independente não tem impacto sobre o esforço de desenvolvimento.

Neste sentido, foram criadas estruturas de serviços por módulos, e a granularidade dos micro serviços foi definida como já apresentado anteriormente. Desse modo, ainda assim, mantém em concordância com os principais princípios, como por exemplo, o modelo em torno de conceitos de negócios. Também, outro ponto que pode impactar negativamente no uso da abordagem de criar toda uma estrutura para cada micro serviço é em relação ao aumento considerável de memória e, consequentemente, os custos elevados com servidores que a empresa deverá arcar para manter sua solução no ar.

Na Figura 4, é apresentada uma parte de como foram construídos os micro serviços, em destaque para o módulo tributário, que foi o módulo principal de estudo do experimento. Como pode ser observado, foram criados *web services* para cada módulo: WS Container Tributário, WS Container Cadastro, WS Container Financeiro, e *web services* para os demais módulos. Em relação às funcionalidades da *feature* Tributação, os micro serviços foram agrupados em suas principais *features*, como por exemplo: Na tributação, foram criados os micro serviços referentes às operações de manipulação de registros (MS Consulta, MS Deletar, MS Novo, MS Edição), e as demais regras necessárias para operacionalizar a funcionalidade por completo (MS Get CST, MS Get Situação Tributária ICMS, MS Get Situação Tributária IPI, MS Get Situação Tributária PIS, MS Get Situação Tributária COFINS e MS Get Valida Regras); e da natureza de operação, que também possui os micro serviços referentes a manipulação dos dados, e os micro serviços referentes às suas regras específicas (MS Get CFOP, MS Get Finalidade Emissão e MS Valida Regras).

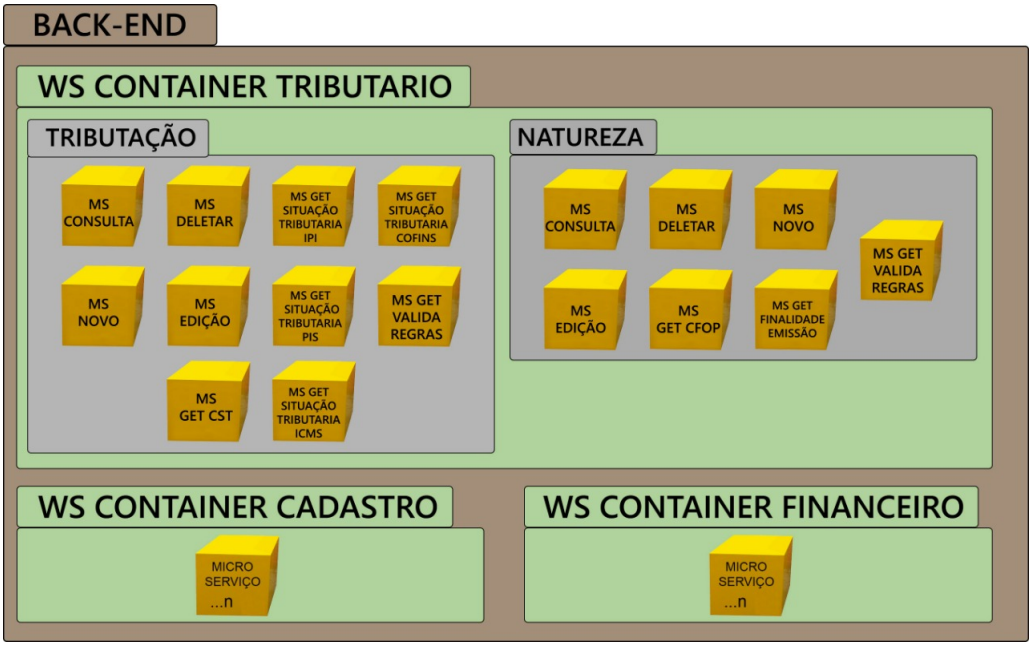

Figura 4 – Parte dos micro serviços construídos

Por fim, quanto a esse processo de migração, acredita-se que ao trabalhar com SPL e micro serviços, o principal desafio está na granularidade das *features*, além da questão de diminuir o acoplamento e coesão de um código existente de uma abordagem tradicional para um código como serviço.

4.2 Estudo de Caso

A avaliação do impacto de micro serviços na abordagem de SPL se deu através da execução de um estudo de caso na Empresa X, com base em dois cenários (com e sem micro serviços), conforme mencionado no início deste capítulo.

Para realizar o planejamento deste estudo de caso foi levado em consideração o processo estabelecido por Wohlin et al. (2012), conforme mostrado na Figura 5. Foi utilizado esse processo pois é comumente encontrado em trabalhos relacionados a Engenharia de Software. Além disso, a escolha em realizar um estudo de caso se deu por ser uma estratégia mais simples e não tão controlada, pois como foi executado em um ambiente de trabalho empresarial, existe a dificuldade em conseguir utilizar os recursos da empresa e parar seus funcionários para realizarem os experimentos.

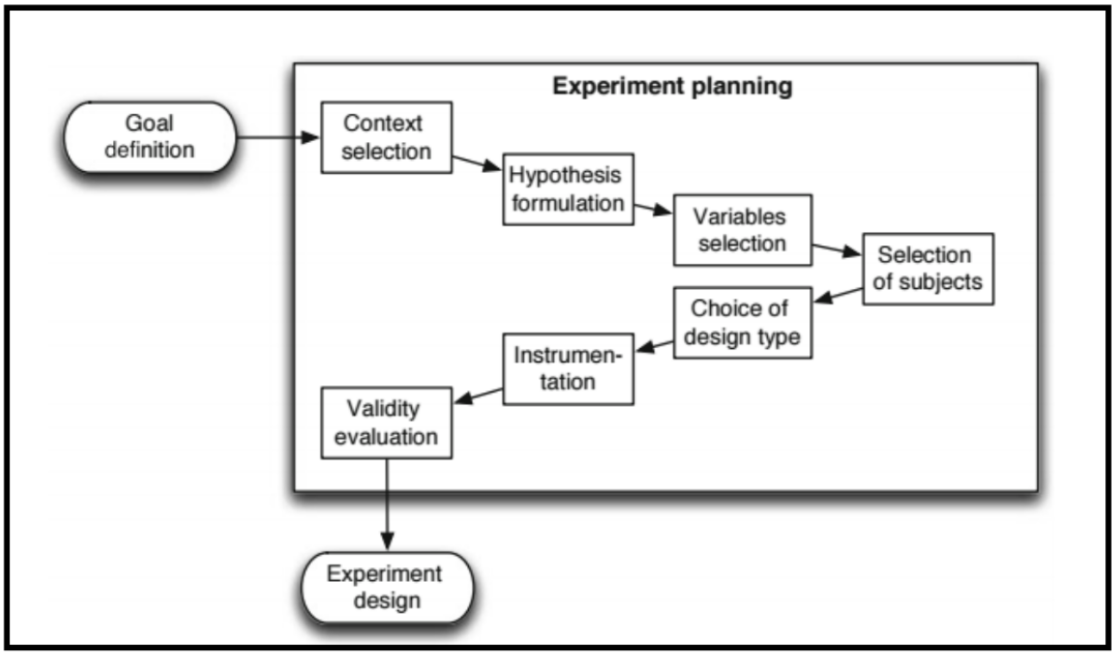

Figura 5 – Processo Wohlin

4.2.1 Objetivos

A primeira etapa deste processo é a definição dos objetivos. Com isso, o experimento se deu com o seguinte objetivo:

Analisar o desenvolvimento de uma SPL,

com o propósito de medir o impacto de micro de serviços,

com relação às tarefas de manutenção e evolução,

do ponto de vista do engenheiro de software,

e no contexto de software de gestão financeira.

Também, a fim de realizar a base de comparação entre a evolução dos dois cenários, foi utilizada a técnica Goal-Question-Metric (GQM), que auxilia na definição e interpretação de métricas de software (FUGGETTA et al., 1998). A seguir são apresentadas as duas questões e métricas estabelecidas.

- **Q1: Qual abordagem demanda mais esforço para realizar as tarefas de adição e evolução de features?**

  Métricas:

  - **M1:** Número de linhas de códigos modificadas.
  - **M2:** Número de arquivos modificados.

  - **M3:** Número de horas para realizar a mudança.

  Para as métricas M1 e M2 foi utilizado o repositório *GIT* para coletar os dados, enquanto que para a M3, foi utilizada uma planilha onde os dois desenvolvedores preencheram conforme a execução do experimento.

- **Q2: Qual abordagem é mais difícil de testar e entregar?**

  Métricas:

  - **M4:** Percepção sobre a dificuldade de testar.
  - **M5:** Percepção sobre a dificuldade para entregar o software.

  Para as métricas M4 e M5, foi feito um questionário que foi respondido pelos desenvolvedores ao final do experimento.

4.2.2 Seleção de contexto, variáveis e hipóteses

Nas próximas etapas, foram definidas a seleção de contexto e a seleção de indivíduos. Como o trabalho tem parceria com a Empresa X, o experimento foi executado em um ambiente empresarial, com base em um dos produtos da empresa, ou seja, foi utilizada uma abordagem *in-vivo*, sendo categorizada como *on-line*. Seguindo essa decisão, o experimento foi executado por engenheiros de softwares da própria empresa X. Para garantir uma análise balanceada, foram selecionados dois engenheiros com a mesma experiência, mesmo *skill* e o mesmo conhecimento do projeto. Além disso, classifica-se este experimento como geral, pois pode ser aplicado a outros tipos de *software* com o uso das mesmas tecnologias.

Outro ponto que foi necessário definir foi a seleção de variáveis. Segundo Wohlin et al. (2012), as variáveis dependentes tem como principal objetivo descrever os valores que representam o desempenho e a eficácia das abordagens testadas. Nesse sentido, para o nosso experimento, foram definidas as seguintes variáveis dependentes: quantidade de código fonte alterado, considerando-se os *branchs* do repositório *GIT*, esforço gasto na realização das tarefas, medido pela planilha para medição do tempo de trabalho, e a percepção sobre a dificuldade de se testar e implantar o software, medida por meio das respostas no questionário de avaliação. Já as variáveis independentes consistem nas variáveis que auxiliam na avaliação dos efeitos do experimento. Com isso, foram definidas as seguintes variáveis independentes: arquitetura do código fonte, com e sem micro serviço, e as tarefas descritas em formato de casos de usos.

Quanto à formulação das hipóteses, foram definidas três perguntas a fim de conseguir refutar a hipótese nula de cada, conforme mostrado a seguir. Cada questão está diretamente relacionada com uma métrica da Q1, ou seja, a QP1 está relacionada com a

M3, a QP2 está relaciona com as métricas M1 e M2, e por fim, a QP3 está relacionada com as métricas M4 e M5.

- **QP1: Qual abordagem possibilita menor alteração em código para as tarefas de evolução da SPL?**
  - H_0**:** $\Phi_{at} = \Phi_{ams}$ = A abordagem tradicional altera o mesmo número do que na abordagem de micro serviços.
  - H_1**:** $\Phi_{at} > \Phi_{ams}$ = A abordagem tradicional altera mais código que a abordagem de micro serviços.
  - H_2**:** $\Phi_{at} < \Phi_{ams}$ = A abordagem tradicional altera menos código que a abordagem de micro serviços.
- **QP2: Qual abordagem requer menos tempo de desenvolvimento para a realização das tarefas de evolução da SPL?**
  - H_0**:** $\Phi_{at} = \Phi_{ams}$ = A abordagem tradicional leva o mesmo tempo que na abordagem de micro serviços.
  - H_1**:** $\Phi_{at} > \Phi_{ams}$ = A abordagem tradicional leva mais tempo que a abordagem de micro serviços.
  - H_2**:** $\Phi_{at} < \Phi_{ams}$ = A abordagem tradicional leva menos tempo que a abordagem de micro serviços.
- **QP3: Qual a abordagem tem maior dificuldade de testar e entregar o software?**
  - H_0**:** $\Phi_{at} = \Phi_{ams}$ = A abordagem tradicional tem a mesma dificuldade que a abordagem de micro serviços.
  - H_1**:** $\Phi_{at} > \Phi_{ams}$ = A abordagem tradicional é mais difícil que a abordagem de micro serviços.
  - H_2**:** $\Phi_{at} < \Phi_{ams}$ = A abordagem tradicional é mais fácil que a abordagem de micro serviços.

4.2.3 Planejamento e execução

Em relação à fase de planejamento do experimento, foi projetada a execução de 8 sessões de uma hora e meia para quatro tarefas. Segundo Pressman (2005), a tarefa de manutenção é classificada em quatro tipos: Manutenção Corretiva, que é a correção de erros no software que não foram identificados na fase teste, a Manutenção Adaptativa, que são as mudanças ocorridas em seu ambiente externo, a Manutenção Evolutiva, que são as alterações não previstas no documento de requisito original do software, e a Manutenção

Preventiva ou Reengenharia, que são as alterações no software buscando fornecer uma estrutura melhor para futuras manutenções. Diante disso, tentamos cobrir esses quatro tipos de tarefas de manutenção, porém devido ao tempo e disponibilidade dos recursos da empresa e levando em consideração que a manutenção preventiva está fora do escopo do trabalho, pois se trata de tarefas de reengenharia de software, as quatro tarefas executadas no experimento foram referentes as tarefas de manutenção evolutiva e adaptativa, duas de cada, conforme apresentado na Tabela 1. Também, nessa etapa de planejamento, foi disposto uma aleatoriedade nas ordens das tarefas e sem agrupamento, e seguindo os seguintes critérios de balanceamento:

- Dois desenvolvedores fazem cada tarefa duas vezes, com e sem micro serviços.
- Cada desenvolvedor começa duas tarefas com micro serviço e depois refaz sem micro serviço. Começa duas tarefas sem micro serviço e depois refaz com micro serviço.
- A ordem com/sem micro serviço é invertida entre os dois desenvolvedores.
- As tarefas ficam espaçadas para evitar de fazer a mesma tarefa em sequência.

Tabela 1 – Tarefas Experimento

Manutenção Evolutiva	**Manutenção Adaptativa**
T01 - Cadastro de Tributação (ICMS e IPI)	T01-1 - Incluir PIS, COFINS na Tributação
T02 - Cadastro de Natureza de Operação	T02-1 - Incluir Impostos na Natureza de Operação

Sendo assim, a simulação das tarefas executadas ficaram conforme mostrado na Tabela 2.

Tabela 2 – Simulação Experimento

Sessão	**Desenvolvedor A**	**Desenvolvedor B**
1	T01 - Sem MS	T01 - Com MS
2	T02 - Com MS	T02 - Sem MS
3	T01-1 - Sem MS	T01-1 - Com MS
4	T02-1 - Com MS	T02-1 - Sem MS
5	T01 - Com MS	T01 - Sem MS
6	T02 - Sem MS	T02 - Com MS
7	T01-1 - Com MS	T01-1 - Sem MS
8	T02-1 - Sem MS	T02-1 - Com MS

Não foi incluído o *deploy*/entrega do software ao cliente, pois queríamos focar mesmo no desenvolvimento, e não na entrega. A expectativa era que a abordagem de micro serviço tornaria essa entrega instantânea, mas isso já é bem conhecido, como apresentado no trabalho de Tizzei et al. (2017), portanto fora do foco deste trabalho.

Como instrumento deste experimento, foram disponibilizados casos de uso das tarefas para os indivíduos, cópias do código fonte dos dois cenários já configurados no *GIT*, apresentado todo o processo do experimento, além de disponibilizar as máquinas para desenvolvimento com o todos os programas já configurados. A fim de mapear os dois desenvolvedores que executariam o experimento e manter os pré requisitos estabelecidos, foi aplicado um questionário para mapear conhecimento, o qual é apresentado na Figura 6. Também, foi disponibilizada uma planilha para documentar as horas trabalhadas em cada tarefa e vincular o código do *commit*. Por fim, com base nos artefatos do software (código fonte) gerados, foi utilizada a ferramenta *GIT* para conseguir posteriormente fazer a análise e extração das métricas pré definidas. Para isso, foi necessário, a cada sessão do experimento, realizar a cópia do projeto, criar um *branch* novo e no final realizar o *commit* com o trabalho desenvolvido, para cada indivíduo e tarefa.

Questionário Mapeamento do Conhecimento		
Questão (escala de 0 a 5)	**A**	**B**
Trabalhou com micro serviço	4	3
Trabalhou com Spring Boot	4	4
Trabalhou com SPL	4	4
Experiência com Java	5	5
Experiência como Desenvolvedor	5	5
Experiência escopo tributário do ERP	5	4
Nota Total	**27**	**25**

Figura 6 – Questionário Mapeamento do Conhecimento

Com todo o planejamento realizado, foram apresentados aos dois desenvolvedores, na etapa de treinamento, o processo da execução do experimento (conforme mostra a Figura 7), que foi realizado em duas sessões de uma hora e meia antes do início do experimento. Neste processo, é dado como entrada os casos de uso das tarefas e os código fonte (tanto do cenário com micro serviços, como do cenário sem micro serviços), e a cada início da sessão o desenvolvedor devia criar um *branch* e registrar hora de início da atividade. Caso ocorresse alguma pausa que não estava no planejamento, os desenvolvedores anotavam o horário que paravam e ao voltar realizavam a anotação dos horários na planilha, assim evitando considerar tempo do experimento com tarefas não relacionadas ao objetivo do mesmo. Após realizar a tarefa durante o horário da sessão, o desenvolvedor finalizava o registro do horário na planilha e submetia seu *branch*. Por fim, tínhamos o projeto evoluído para análise posterior dos dados do experimento com auxílio do GIT. Vale destacar que esse processo foi repetido em oito sessões de uma hora e meia para quatro tarefas, nos cenários com e sem micro serviços.

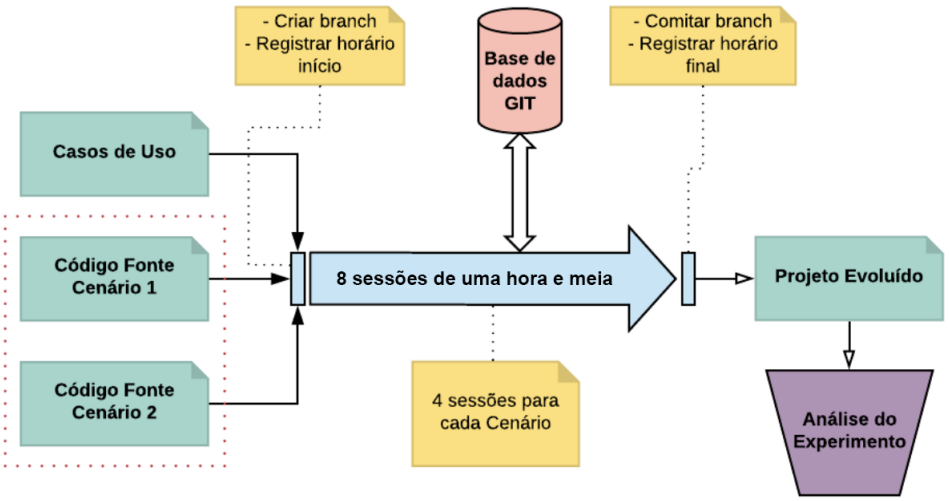

Figura 7 – Visão Geral do Processo do Experimento

4.2.4 Resultados e discussão

Após a execução do experimento, e com os dados coletados, procedeu-se à sua análise e discussão, visando comparar o que foi observado com as hipóteses formuladas e com outros resultados obtidos na literatura.

Uma primeira observação a ser feita é que, em todas as tarefas, os desenvolvedores utilizaram a janela de 1h30min de cada sessão em sua totalidade. Uma vez que a corretude do que foi entregue foi verificada em todos os casos, isso significa que não foi observada diferença de tempo no experimento. Mas isso não significa que ambas abordagens levam a um mesmo tempo. Como a janela era fixa, pode ser que alguns desenvolvedores permaneceram algum tempo ociosos ou verificando o que tinha sido feito até o tempo se esgotar, aumentando o tempo artificialmente em uma ou outra abordagem. Essa foi uma limitação do experimento, como discutido na Seção 4.2.5.

A Figura 8 e a Figura 9 apresentam a compilação dos dados extraídos do *GIT*. Nessas duas figuras é apresentado o levantamento por desenvolvedor e por tipo de tarefas avaliadas.

Analisando-se a questão de pesquisa QP2 (Qual abordagem requer menos tempo de desenvolvimento?), formulada na Seção 4.2.2, concluímos que a hipótese nula não foi refutada, ou seja, não foram observados indícios de que a abordagem de micro serviços aumenta ou diminui o tempo de entrega do software.

Resultados Experimento							
Tarefa	Tipo	Desenvolvedor	Total Horas	Arquivos Modificados	Linhas Adicionadas	Linhas Removidas	Linhas Modificadas
Cenário Sem Micro Serviço							
T01 - Sem MS	Man. Evolutiva	A	01:30	15	1116	0	1116
		B	01:30	7	699	0	699
T01-1 - Sem MS	Man. Adaptativa	A	01:35	14	562	42	604
		B	01:30	15	231	7	238
Total		**A**	**03:05**	**29**	**1678**	**42**	**1720**
		B	**03:00**	**22**	**930**	**7**	**937**
T02 - Sem MS	Man. Evolutiva	A	01:38	9	512	0	512
		B	01:30	10	383	0	383
T02-1 - Sem MS	Man. Adaptativa	A	01:30	5	424	66	490
		B	01:30	15	586	90	676
Total		**A**	**03:08**	**14**	**936**	**66**	**1002**
		B	**03:00**	**25**	**969**	**90**	**1059**

Figura 8 – Resultado Experimento - Cenário Sem Micro Serviço

Resultados Experimento							
Tarefa	Tipo	Desenvolvedor	Total Horas	Arquivos Modificados	Linhas Adicionadas	Linhas Removidas	Linhas Modificadas
Cenário Com Micro Serviço							
T01 - Com MS	Man. Evolutiva	A	01:31	50	1332	0	1332
		B	01:30	38	873	0	873
T01-1 - Com MS	Man. Adaptativa	A	01:27	8	180	1	181
		B	01:30	8	167	2	169
Total		**A**	**02:58**	**58**	**1512**	**1**	**1513**
		B	**03:00**	**46**	**1040**	**2**	**1042**
T02 - Com MS	Man. Evolutiva	A	01:29	15	1081	0	1081
		B	01:30	11	1294	0	1294
T02-1 - Com MS	Man. Adaptativa	A	01:31	1	172	3	175
		B	01:30	2	306	5	311
Total		**A**	**03:00**	**16**	**1253**	**3**	**1256**
		B	**03:00**	**13**	**1600**	**5**	**1605**

Figura 9 – Resultado Experimento - Cenário Com Micro Serviço

A Figura 10 mostra um comparativo das tarefas de manutenção evolutiva. Como pode ser observado, os desenvolvedores modificaram muito mais arquivos na abordagem com micro serviços.

A explicação para essa observação é que, como a tarefa evolutiva é uma tarefa de manutenção onde serão adicionadas novas *features* na linha de produto, quando se utiliza micro serviços é necessário criar uma série de arquivos que a estrutura exige (como por exemplo, as classes *"resource"*, *"filter"*, *"repository"* e *"service"*, apresentadas na seção 4.1), principalmente quando se trabalha com persistência de dados. Isso não ocorre por exemplo, na arquitetura sem micro serviço, e por isso há menor necessidade de se adicionar arquivos. Em termos de linhas de código modificadas, também houve um maior

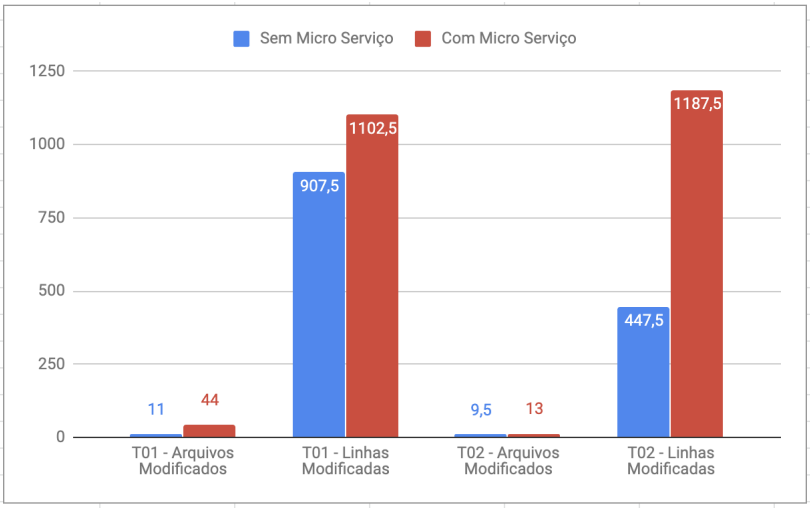

Figura 10 – Comparativo tarefa de manutenção evolutiva - Média por Desenvolvedor

número de linhas de códigos modificadas no cenário com micro serviço, como apresentado na Figura 10.

Se nas tarefas evolutivas observou-se que o uso de micro serviços pode levar a um maior esforço em termos de número de arquivos e linhas de código alterados, nas tarefas adaptativas o esforço extra é compensado. Conforme mostra a Figura 11, nas quatro sessões observou-se que as tarefas realizadas nos cenários com micro serviços exigiram a modificação de um número menor de arquivos e de linhas de código. De acordo com os dados do experimento realizado, a estruturação do código em um projeto mais distribuído em menores fragmentos de código aumenta a coesão e facilita a realização de adaptações.

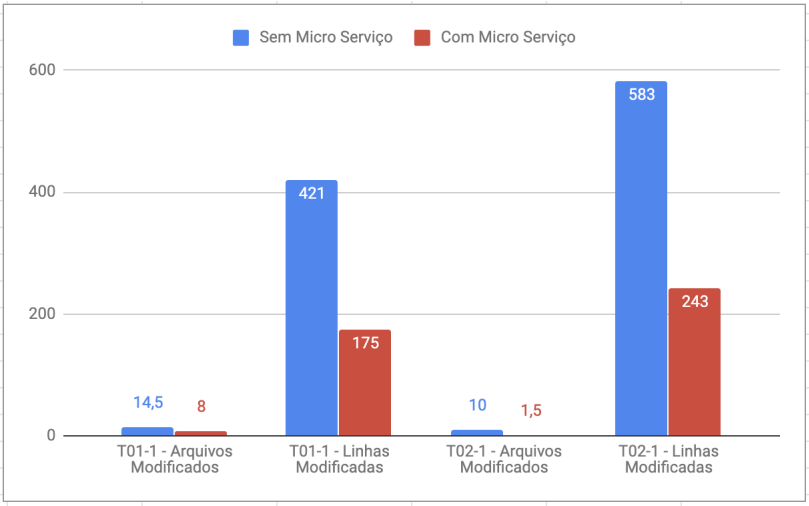

Figura 11 – Comparativo tarefa de manutenção adaptativa - Média por Desenvolvedor

Retornando à questão de pesquisa QP1 (Qual abordagem possibilita menor alteração em código?), observamos indícios de que a hipótese nula é refutada. Dependendo

da natureza da tarefa, o uso de micros serviços pode aumentar ou diminuir o esforço necessário para manutenção.

Por fim, após a realização do experimento, foi solicitado aos dois desenvolvedores que respondessem quatro questões sobre o trabalho desenvolvido, que são:

- **1) Qual abordagem gera mais dificuldade em realizar teste?**
  - **Desenvolvedor A:**

    Resposta: Abordagem Tradicional

    "A abordagem tradicional necessita da criação de testes unitários. Quando utilizamos testes unitários a injeção de dependências, principalmente relacionadas a persistência, dificulta os testes. Em contra ponto, os micro serviços podem ser testados em unidades pequenas através de ferramentas de consumo de web serviços, como por exemplo via Postman (IDE para testes de serviços). Essas ferramentas facilitam muito o teste das funcionalidades, permitindo saber se a funcionalidade está funcionando como esperado antes da criação do *front-end*."
  - **Desenvolvedor B:**

    Resposta: Abordagem Tradicional

    "Devido às bibliotecas para testes não funcionarem 100% com o *framework* utilizado na abordagem tradicional."
- **2) Qual abordagem tem o melhor processo de entrega de software?**
  - **Desenvolvedor A:**

    Resposta: Abordagem Tradicional

    "A abordagem tradicional permite entregar mais rápido pequenos incrementos no software, pois tem uma quantidade menor de arquivos necessários a serem criados para implementar a funcionalidade."
  - **Desenvolvedor B:**

    Resposta: Abordagem Tradicional

    "Devido a utilização de ORM para a criação das tabelas do banco, e não possuir tantas classes para se trabalhar com o banco de dados se torna mais rápido o processo na abordagem tradicional."
- **3) Qual abordagem permite evoluir um software com mais qualidade (código)?**
  - **Desenvolvedor A:**

    Resposta: Micro Serviço

"Micro serviços se mostraram melhores de realizar a manutenção ou evolução, visto que como são implementados em unidades pequenas diminui o acoplamento entre as camadas, e também facilita o entendimento do código."

- **Desenvolvedor B:**

  Resposta: Micro Serviço

  "Na abordagem de micro serviços pelo software ficar mais modular se torna mais fácil a manutenção de partes, e os testes também são mais fáceis de serem realizados, fazendo com que em uma manutenção fique mais fácil de detectar erros e de se consertar."

- **4) Cite as principais dificuldades em executar cada abordagem**

  - **Desenvolvedor A:**

    "Micro serviços se mostraram mais custosos de serem implementados, pois necessitam de mais classes para serem implementados, diferente da abordagem tradicional. Além disso, facilitou bastante a manutenção e o entendimento do código na abordagem usando micro serviço. Além disso, pensando além do código *front-end*, para web, os serviços podem ser consumidos por diferentes dispositivos."

  - **Desenvolvedor B:**

    "Na abordagem tradicional a principal dificuldade é a realização de teste, e a dependência entre as bibliotecas.

    Na abordagem de micro serviços, a quantidade de classes necessárias para se trabalhar com o banco de dados."

Analisando-se as respostas dos desenvolvedores, a abordagem tradicional gerou uma percepção de dificuldade maior quando o assunto foi realizar teste, devido à sua compatibilidade com os *frameworks* utilizados no experimento, enquanto que a abordagem de micro serviço possui algumas ferramentas (como por exemplo, o *Postman*) que facilitam a automatização de teste de serviço. Também, os desenvolvedores mencionaram no questionário que a abordagem tradicional facilita a entrega e torna esse processo mais rápido pois existe uma estrutura menor a ser modificada. Por outro lado, escolheram a abordagem de micro serviço quando a avaliação é evoluir um software com mais qualidade (código), devido à estruturação em partes menores e mais bem separadas, cada uma com sua responsabilidade, o que diminui o acoplamento e consequentemente uma evolução com mais qualidade. Sendo assim, na questão de pesquisa QP3 (Qual a abordagem tem maior dificuldade de testar e entregar o software?) a hipótese nula não pode ser refutada, uma vez que cada abordagem levou alguma vantagem em cada avaliação da questão.

Ainda segundo levantamento teórico e confirmado no trabalho de Tizzei et al. (2017), a utilização de SPL e micro serviços traz grandes benefícios para sistemas que

desejam conseguir uma maior escalabilidade e melhora na manutenção do software. No nosso trabalho, diante do experimento realizado, conseguimos avaliar que a utilização de micro serviços traz grandes benefícios na evolução do software, principalmente nas tarefas adaptativas. Esse benefício é comprovado uma vez que nas tarefas adaptativas houve uma redução de 58,37% de linhas de código e 61,22% de redução de arquivos modificados do que a abordagem tradicional. Por outro lado, nas tarefas evolutivas micro serviços não teve uma boa análise quando comparado com a abordagem tradicional. Isso reflete no grande aumento de número de linhas de códigos modificados e arquivos modificados, onde teve um acréscimo de 69% e 178,05% respectivamente.

4.2.5 Ameaças à validade

Quanto às limitações deste trabalho, podemos citar por primeiro que o experimento foi realizado com poucas pessoas, no nosso caso dois desenvolvedores, e executado com poucas tarefas. Porém, como estava sendo consumido tempo e recursos de uma empresa, é difícil conseguir realizar esse tipo de atividade com mais pessoas ou com um número maior de tempo como desejável. Em segundo lugar, o tempo dos experimentos ficou limitado. Isso pode ter eliminado a variável "tempo" da análise, porém, todas as tarefas foram verificadas e completadas corretamente, por mais que tenham ficados com tempos parecidos. Essa limitação também surgiu do fato de o experimento estar sendo realizado em uma empresa. Foi difícil negociar o uso de recursos por um tempo ilimitado, portanto o uso de sessões com uma duração fixa foi um compromisso aceitável.

Em terceiro lugar, não testamos as tarefas de manutenção classificadas como corretiva e preventiva. A última estava fora do escopo de trabalho, por se tratar de tarefas de reengenharia de software. Já a corretiva, não foi possível avaliá-la uma vez que faltou tempo por parte da Empresa X, mas a expectativa era que a abordagem de micro serviço teria uma melhor performance do que a abordagem sem micro serviço, seguindo a mesma justificativa da manutenção adaptativa apresentada anteriormente.

Abaixo são apresentadas as principais ameaças à validade do resultado e suas análises:

- **Validade de Conclusão: O tratamento introduzido teve um efeito que pode ser significativamente observado (de preferência com análise estatística) na saída?** Não fizemos análise estatística devido ao baixo número de participantes, mas os resultados indicam claramente que nas tarefas evolutivas, no cenário 2, foi necessário um maior número de arquivos/linhas de código, e nas tarefas adaptativas foi necessário um menor número de arquivos/linhas de código. Em termos de tempo das tarefas, não foi observada diferença significativa.

  - **Confiabilidade das medidas:** Com a garantia do total conhecimento do escopo, através da experiência de quem gerenciará o experimento, as mesmas informações foram passadas a equipe de desenvolvimento. Por fim, foi solicitado a entrega do artefato de software (código fonte) submetido no *GIT*.
  - **Irrelevâncias na configuração experimental:** Não foi permitido máquinas pessoais, sendo disponibilizadas todas as máquinas pela empresa, previamente configuradas com todos os recursos necessários ao desenvolvimento.
  - **Heterogeneidade dos participantes:** Foi realizado um questionário para mapeamento do conhecimento, a fim de realizar o alinhamento de todos os envolvidos na fase de treinamento.

- **Validade Interna: O tratamento introduzido foi de fato a causa da saída observada? Ou existem outros fatores que podem ter influenciado?** Como o estudo de caso foi feito em um ambiente não 100% controlado, podem existir outros fatores de influência, mas tentou-se limitar as tarefas a sessões fechadas, sem que os participantes pudessem estar fazendo outras tarefas em paralelo. Além disso, as mesmas tecnologias foram mantidas, restringindo as mudanças ao tratamento apenas (uso de micro serviços). Existe ainda o viés de aprendizado, que é um fator que influencia apenas na primeira vez, porque depois que a equipe tiver dominado as tecnologias, não terá mais influência. Para o experimento, essa questão foi mitigada através de um treinamento anterior à execução do mesmo.

- **Validade de Construção: O tratamento corresponde de fato à causa que estamos interessados? A saída corresponde de fato ao efeito que estamos interessados?** O tratamento oferecido no cenário dois mistura conceitos de SPL com micro serviços, em contraste com o cenário um, que não emprega nenhum tipo de gerenciamento de SPL. Assim, a saída pode estar sendo observada não exclusivamente devido ao uso de micro serviços, mas de forma combinada com o gerenciamento de SPL introduzido. Idealmente, deveriam ser avaliados quatro cenários: os cenário um e dois descritos, um cenário três, onde apenas se introduziu o gerenciamento de SPL, e um cenário quatro, onde apenas foram utilizados micro serviços. Nessa construção, poderia-se isolar os efeitos de cada abordagem. Porém, tal avaliação seria muito difícil de ser alcançada, uma vez que o projeto correto dos micro serviços, seu escopo e dimensão, exigiu uma análise do domínio mais completa, característica de SPL. Além disso, essa experimentação exigiria o dobro do esforço dos pesquisadores e da empresa, que neste momento não foi possível negociar. Assim, decidiu-se avaliar o efeito da adoção conjunta de SPL e micro serviços, desprezando-se os efeitos individuais, uma vez que os benefícios exclusivos de SPL e micro serviços já são mais bem conhecidos.

Também, outro fator que podia influenciar era dos participantes acharem que estavam sendo avaliados. Sendo assim, deixamos sempre claro que estávamos avaliando as abordagens utilizadas e não o trabalho deles em relação a empresa.

Outra ressalva à validade de construção diz respeito ao tempo de cada tarefa. As sessões foram limitadas a 1h30min cada, o que de certa forma obrigou que os tempos ficassem similares em todas as tarefas. Sem restrição de tempo, pode ser que as tarefas tivessem tempos mais variados.

- **Validade externa ou transferibilidade: A relação causa-efeito demonstrada é válida para outras situações? Podemos generalizar os resultados? Os resultados são aplicáveis em outros contextos?** Os resultados devem ser parecidos para outros contextos dentro da similaridade com os cenários testados: uma linha de produto gerenciada de forma *ad-hoc*, utilizando tecnologias web, em um domínio de ERP com uma família de produtos de tamanho/escopo similares a este. Acredita-se que resultados similares sejam observados em outros domínios e em outras empresas, mas sem análises adicionais não é possível garantir essa validade ainda.

- **Credibilidade: Estamos confiantes de que as observações são verdadeiras? Por quê?** Dentro do que foi possível controlar e observar, não foi observado nenhum problema que levantasse suspeita nas observações. O experimento foi acompanhado o tempo todo pelo pesquisador.

- **Dependabilidade: As observações são consistentes? Elas podem ser repetidas?** As observações são condizentes com a intuição, com a observação do código-fonte, e com o cruzamento dos dados coletados das diferentes métricas. As diferenças nas quantidades de código alterados em cada tarefa de fato refletem o que é necessário em termos de codificação para cada cenário.

4.3 Considerações Finais

Neste capítulo apresentou-se a avaliação conduzida nesta pesquisa, desde o planejamento do experimento até a execução e compilação dos resultados obtidos.

Primeiro, vimos todo o esforço em migrar um projeto desenvolvido na abordagem tradicional com SPL para arquitetura de micro serviços, criando assim o cenário 2. Em termos de tamanho dos micro serviços, como não existe um consenso de tamanho ideal, ainda é necessário que o engenheiro de software tenha uma certa experiência com base em cada produto que será projetado por essa abordagem, tendo como aliadas nessa migração as diretrizes criadas por Newman (2015), Lewis e Fowler (2014).

Em seguida, foi feito todo um planejamento e execução de um experimento utilizando o processo estabelecido por Wohlin et al. (2012), em cima de dois cenários (um com micro serviços e outro sem micro serviços) de uma linha de produto de uma empresa parceira deste trabalho. Através disso, conseguimos avaliar que a utilização de micro serviços pode trazer benefícios na evolução de uma linha de produto de software, principalmente nas tarefas adaptativas, que são as mudanças ocorridas em seu ambiente externo ao da aplicação. Além disso, como avaliação dos desenvolvedores envolvidos no experimento, conclui-se que a abordagem tradicional tem o melhor processo de entrega de software, porém gera mais dificuldade em realizar teste. Em contra partida, micro serviços ganham no quesito evoluir um software com mais qualidade, conforme percepção relatada pelos desenvolvedores.

Por fim, nosso trabalho apontou alguns pontos de ameaças à validade do mesmo, como por exemplo, a execução do experimento com poucos desenvolvedores e em poucas tarefas, porém como executamos o experimento em um ambiente empresarial, existe a dificuldade em conseguir parar e utilizar os recursos da empresa. Outro ponto mencionado foi a questão de não incluir o *deploy*/entrega do software ao cliente. A expectativa era que a abordagem de micro serviço tornasse essa entrega instantânea, mas isso já é bem conhecido, como apresentado no trabalho de Tizzei et al. (2017), portanto fora do foco do nosso trabalho.

5 Considerações Finais e Trabalhos Futuros

Como bem se sabe, com o avanço das tecnologias atuais e a alta demanda de software como serviço, cada vez mais as empresas estão se vendo obrigadas a modernizar suas aplicações para o mundo digital. No entanto, essa modernização precisa ser muito bem projetada para conseguir acompanhar as necessidades dos clientes e também manter um custo baixo de operação para a empresa, principalmente em aplicações que possuem uma grande variabilidade de funcionalidades para seus clientes. Nesse contexto, um dos problemas enfrentados é em relação ao gerenciamento de múltiplas versões de sistemas "semelhantes" e da evolução do software. Para resolver a questão do gerenciamento de múltiplas versões utiliza-se a abordagem de Linha de Produto de Software (SPL), enquanto que o estilo arquitetural de Micro Serviço é uma forma de conseguir deixar o processo de evolução mais leve, uma vez que as alterações ficam confinadas em unidades menores e mais fáceis de serem gerenciadas e implantadas. Enquanto os benefícios da arquitetura de micro serviços são bem conhecidos em termos de operação e implantação, seu uso em uma linha de produto , do ponto de vista da Engenharia de Software, não é objeto de muitos estudos na literatura. Com isso, esta pesquisa de mestrado teve como principal objetivo avaliar a evolução (tanto na adição de novas *features*, como na manutenção de *features* existentes) de uma SPL, através da utilização de micro serviços, para no final conseguir avaliar se o impacto de micro serviços realmente melhorou a tarefa de evolução. Além disso, os autores Tizzei et al. (2017) deixaram como trabalhos futuros avaliar o uso de micro serviços na perspectiva da evolução do software, sendo esta a motivação do presente trabalho.

A principal contribuição deste trabalho foi a observação de como a arquitetura de micro serviços impactou nas tarefas de evolução de uma linha de produtos, sob o ponto de vista da Engenharia de Software. A redução significativa no número de arquivos/linhas de código alterados nas tarefas adaptativas mostra que o potencial para redução de esforço existe e deve ser explorado. Também mostra que esse benefício vem com um custo, uma vez que realizar a migração para essa arquitetura e o processo de adicionar novas *features* pode exigir mais esforço. A quantificação exata do custo X benefício pode ser objeto de estudo de trabalhos futuros, auxiliando no processo de tomada de decisão quanto à migração ou não para essa arquitetura.

Além da avaliação do impacto do uso de micro serviços na evolução de uma linha de produto, este trabalho contribui com as lições aprendidas da migração de um projeto desenvolvido na abordagem tradicional para o uso de micro serviços, utilizando dados de um experimento real com foco na Engenharia de Software. As decisões de projeto ajudam a validar as diretrizes criadas por Newman (2015) e Lewis e Fowler (2014). As seguintes

lições aprendidas podem ser destacadas:

- Uma vez que não há consenso sobre o tamanho e escopo de um micro serviço, a experiência dos desenvolvedores é um fator importante na decisão de projeto dos micro serviços.
- Outro fator importante ao projetar uma linha de produto utilizando micro serviços é sempre ter uma visão geral de toda a linha, pois assim, será mais fácil de conseguir projetar o nível de granularidade de cada micro serviço. Para isso, utiliza-se a modelagem de *feature* (feature model), que permite modelar todas as variabilidade da SPL.
- Separar os micro serviços primeiramente por módulos, e depois em níveis de operações de manipulação de registro e os demais micro serviços conforme as pequenas regras de cada contexto, foi uma decisão que consideramos acertada, pois assim os micro serviços foram modelados em torno de conceitos de negócios, tornaram-se descentralizados, ocultando detalhes de implantação internas e a implantação pode ser independente, conforme recomendam as diretrizes de Newman (2015), Lewis e Fowler (2014).
- Por outro lado, visando um dos principais pontos negativos de trabalhar com micro serviço, que é o fato dos serviços não guardarem nenhum estado, ou seja, cada requisição terá um estado que será apagado quando a requisição terminar, sendo necessário enviar junto a cada requisição várias informações de dados de acesso e outras informações necessárias para realizar a operação. Esse ponto pode tornar o desenvolvimento um pouco complicado para quem é acostumado a trabalhar na abordagem tradicional, porém após algum tempo de trabalho esse processo fica mais natural.

Como trabalhos futuros, recomendamos um estudo mais longitudinal, mais ao longo do tempo no processo de evolução de uma linha de produto com micro serviços. Espera-se que ao longo do tempo a equipe ganhe mais experiência e com isso use melhor a arquitetura, detecte problemas mais facilmente, entre outros fatores positivos advindos da experiência. Além disso, ao longo do tempo será possível perceber benefícios de longo prazo, como a questão do versionamento com mais clientes (combinações de *features* que não existem hoje), além de que, a linha de produto estará preparada e estruturada da melhor forma para realizar qualquer tipo de evolução.

Também, a realização de um experimento com mais pessoas, mais clientes, mais *features* e abordar os quatro tipos de tarefas conceituadas por Pressman (2005) (Manutenção Corretiva, Manutenção Adaptativa, Manutenção Evolutiva e Manutenção Preventiva ou Reengenharia), seria uma análise interessante a ser trabalhada. Nesse contexto, tendo

uma base maior de teste e abrangendo os quatro tipos de tarefas, consegue-se avaliar estatisticamente, além de conseguir avaliar os dois tipos de manutenção não avaliadas neste trabalho.

A Casos de Uso Utilizados no Estudo de Caso

A seguir é apresentado os casos de uso utilizados no estudo de caso pelos engenheiros de software.

Especificação de Caso de Uso
Sistema

Casos de Uso

1 [UC 01] – Cadastrar Natureza de Operação

1.1 Pré-Condições

O sistema iCode tem que estar no ar e funcionando normalmente;

1.2 Pós-Condição

Uma tela contendo mensagem de sucesso.

1.3 Fluxo Principal

Passos	Ações
1.	O usuário clica no botão “Tributário” no menu do sistema
2.	O sistema exibe as opções relacionadas a opção “Tributário”
3.	O usuário seleciona a opção “Natureza de Operação”
4.	O sistema exibe a tela de cadastro de Natureza de Operação
5.	O usuário seleciona o botão “Novo”

6.	O sistema exibe a tela de cadastro de Natureza de Operações contendo:
	- o Tipo de Operação
	- a Descrição
	- a Finalidade Emissão
	- a Tributação
	- a Conta Gerencial - A vista
	- a Conta Gerencial - A prazo
	- o campo de Natureza exige CFOP específica
	- as opções de Controles:
	  - Calcular valor aproximado dos tributos - Fonte IBPT
	  - Calcular aproveitamento de crédito ICMS
	  - Mostrar mensagem enquadramento federal
	  - Considerar Tabela Alíquota ICMS Interestadual
	  - Destacar ICMS na DANFe
	  - Exige documento referenciado
	  - Gerar duplicata
	  - Incluir IPI no cálculo do ICMS
	  - Movimentar Financeiro

	- as opções de Cálculo de Imposto Estadual:
	  - Calcula ICMS
	  - Calcula ICMS ST
	  - Calcula PIS
	  - Calcula PIS ST
	  - Calcula COFINS
	  - Calcula COFINS ST
	  - Calcula IPI
	
	- as opções de Cálculo de Imposto Interestadual:
	  - Calcula ICMS
	  - Calcula ICMS ST
	  - Calcula PIS
	  - Calcula PIS ST
	  - Calcula COFINS
	  - Calcula COFINS ST
	  - Calcula IPI

7.	O usuário seleciona no campo **Tipo de Operação** a opção “Entrada” e **Finalidade Emissão**: “Normal”
8.	O sistema atualiza as opções de **Controle**, exibindo as opções: - Destacar ICMS (selecionada) - Gerar Duplicata (selecionada) - Movimentar Financeiro (selecionada) - Exige Documento Referenciado (desselecionada) - Incluir IPI no cálculo do ICMS (selecionada)
9.	O usuário: - informa no campo **Descrição**: “Compra de Equipamento” - informa no campo **Tributação**: “6 - Tributação 900” - informa no campo **Natureza exige CFOP específica**: desseleciona - informa no campo **Conta Gerencial - A Vista**: “4.2.01 - Fretes” - informa no campo **Conta Gerencial - A Prazo**: “4.2.01 - Fretes” - informa no campo **CFOP Estadual**: “1.450 Sistemas de Integração”

	- deixa todas as opções de **CFOP Estadual** selecionadas - informa no campo **CFOP Interestadual** “2.101 Compra para Industrialização” - deixa todas as opções de **CFOP Interestadual** selecionadas - clica no botão “Salvar”
10	O sistema emite a mensagem de “Mensagem: Registro salvo com sucesso!” e atribui um código ao cadastro

1.4 Fluxo secundário A

Passos	Ações
7.A.1	O usuário seleciona no campo **Tipo de Operação** a opção “Entrada” e **Finalidade Emissão**: “Ajuste” ou “Complementar”
7.A.2	O sistema atualiza as opções de **Controle**, exibindo as opções: - Destacar ICMS (selecionada) - Gerar Duplicata (selecionada) - Movimentar Financeiro (selecionada) - Exige Documento Referenciado (selecionada) - Incluir IPI no cálculo do ICMS (selecionada)
	Continua no 9 do Fluxo Principal

1.5 Fluxo secundário B

Passos	Ações
7.B.1	O usuário seleciona no campo **Tipo de Operação** a opção “Entrada” e **Finalidade Emissão**: “Devolução”
7.B.2	O sistema atualiza as opções de **Controle**, exibindo as opções: - Destacar ICMS (selecionada) - Gerar Duplicata (selecionada) - Movimentar Financeiro (selecionada) - Exige Documento Referenciado (selecionada e bloqueada pelo sistema) - Incluir IPI no cálculo do ICMS (selecionada)

1.6 Fluxo secundário C

Passos	Ações
8.C.1	O usuário opta por desselecionar a opção “Gerar Duplicata”
8.C.2	O sistema atualiza as opções de **Controle**, excluindo a opção “Movimentar Financeiro”, exibindo as opções: - Destacar ICMS (selecionada) - Gerar Duplicata (desselecionada) - Exige Documento Referenciado (selecionada) - Incluir IPI no cálculo do ICMS (selecionada)
	Continua no 9 do Fluxo Principal

1.7 Fluxo secundário D -- Operação Saída

Passos	Ações
7.D.1	O usuário seleciona no campo **Tipo de Operação** a opção “Saída” e **Finalidade Emissão**: “Normal”
7.D.2	O sistema atualiza as opções de **Controle**, exibindo as opções: - Calcular valor aproximado dos tributos - Fonte IBPT (selecionada) - Calcular aproveitamento de crédito ICMS (selecionada) - Mostrar mensagem enquadramento federal (selecionada) - Considerar Tabela Alíquota ICMS Interestadual (desselecionada e bloqueada pelo sistema) - Destacar ICMS na DANFe (selecionada) - Exige documento referenciado (desselecionada) - Gerar duplicata (selecionada) - Incluir IPI no cálculo do ICMS (selecionada) - Movimentar Financeiro (selecionada)
7.D.3	O usuário: - informa no campo **Descrição**: “Compra de Equipamento” - informa no campo **Tributação**: “6 - Tributação 900” - informa no campo **Natureza exige CFOP específica**: desseleciona

	- informa no campo **Conta Gerencial - A Vista**: “3.1.03.002.001 - Empréstimo” - informa no campo **Conta Gerencial - A Prazo**: “3.1.03.002.001 - Empréstimo” - informa no campo **CFOP Estadual**: “5.101 Venda de Produção do Estabelecimento” - deixa todas as opções de **CFOP Estadual** selecionadas - informa no campo **CFOP Interestadual** “6.000 Saídas ou Prestação de Serviços para Outros Estados” - deixa todas as opções de **CFOP Interestadual** selecionadas - clica no botão “Salvar”
	Continuar no passo 10 do Fluxo Principal

1.9 Fluxo secundário E -- Operação Saída

Passos	Ações
7.E.1	O usuário seleciona no campo **Tipo de Operação** a opção “Saída” e **Finalidade Emissão**: “Ajuste” ou “Complementar”
7.E.2	O sistema atualiza as opções de **Controle**, exibindo as opções: - Calcular valor aproximado dos tributos - Fonte IBPT (selecionada) - Calcular aproveitamento de crédito ICMS (selecionada) - Mostrar mensagem enquadramento federal (selecionada) - Considerar Tabela Alíquota ICMS Interestadual (desselecionada e bloqueada pelo sistema) - Destacar ICMS na DANFe (selecionada) - Exige documento referenciado (selecionada) - Gerar duplicata (selecionada) - Incluir IPI no cálculo do ICMS (selecionada) - Movimentar Financeiro (selecionada)
7.E.3	O usuário: - informa no campo **Descrição**: “Compra de Equipamento” - informa no campo **Tributação**: “6 - Tributação 900” - informa no campo **Natureza exige CFOP específica**: desseleciona - informa no campo **Conta Gerencial - A Vista**: “3.1.03.002.001 - Empréstimo”

	- informa no campo **Conta Gerencial - A Prazo**: “3.1.03.002.001 - Empréstimo” - informa no campo **CFOP Estadual**: “5.101 Venda de Produção do Estabelecimento” - deixa todas as opções de **CFOP Estadual** selecionadas - informa no campo **CFOP Interestadual** “6.000 Saídas ou Prestação de Serviços para Outros Estados” - deixa todas as opções de **CFOP Interestadual** selecionadas - clica no botão “Salvar”
	Continuar no passo 10 do Fluxo Principal

1.9 Fluxo secundário F -- Operação Saída

Passos	Ações
7.F.1	O usuário seleciona no campo **Tipo de Operação** a opção “Saída” e **Finalidade Emissão**: “Devolução”
7.F.2	O sistema atualiza as opções de **Controle**, exibindo as opções: - Calcular valor aproximado dos tributos - Fonte IBPT (selecionada) - Calcular aproveitamento de crédito ICMS (selecionada) - Mostrar mensagem enquadramento federal (selecionada) - Considerar Tabela Alíquota ICMS Interestadual (desselecionada e bloqueada pelo sistema) - Destacar ICMS na DANFe (selecionada) - Exige documento referenciado (selecionada e bloqueada pelo sistema) - Gerar duplicata (selecionada) - Incluir IPI no cálculo do ICMS (selecionada) - Movimentar Financeiro (selecionada)
7.F.3	O usuário: - informa no campo **Descrição**: “Compra de Equipamento” - informa no campo **Tributação**: “6 - Tributação 900” - informa no campo **Natureza exige CFOP específica**: desseleciona - informa no campo **Conta Gerencial - A Vista**: “3.1.03.002.001 - Empréstimo”

	- informa no campo **Conta Gerencial - A Prazo**: “3.1.03.002.001 - Empréstimo” - informa no campo **CFOP Estadual**: “5.101 Venda de Produção do Estabelecimento” - deixa todas as opções de **CFOP Estadual** selecionadas - informa no campo **CFOP Interestadual** “6.000 Saídas ou Prestação de Serviços para Outros Estados” - deixa todas as opções de **CFOP Interestadual** selecionadas - clica no botão “Salvar”
	Continuar no passo 10 do Fluxo Principal

1.8 Fluxo secundário G

Passos	Ações
9.G.1	O usuário opta por desselecionar a opção “Gerar Duplicata”
9.G.2	O sistema atualiza as opções de **Controle**, excluindo a opção “Movimentar Financeiro”, exibindo as opções: - Calcular valor aproximado dos tributos - Fonte IBPT (selecionada) - Calcular aproveitamento de crédito ICMS (selecionada) - Mostrar mensagem enquadramento federal (selecionada) - Considerar Tabela Alíquota ICMS Interestadual (desselecionada e bloqueada pelo sistema) - Destacar ICMS na DANFe (selecionada) - Exige documento referenciado (selecionada e bloqueada pelo sistema) - Gerar duplicata (desselecionada) - Incluir IPI no cálculo do ICMS (selecionada)
	Continua no **7.D.3** do Fluxo Secundário D

2 [UC 02] – Cadastrar Tributação

1.1 Pré-Condições

O sistema iCode tem que estar no ar e funcionando normalmente;

1.2 Pós-Condição

Uma tela contendo todas as informações preenchidas e o sistema sinalizando que a tributação foi salva com sucesso.

1.3 Fluxo Principal

Passos	Ações
1.	O usuário clica no botão “Tributário” no menu do sistema
2.	O sistema exibe as opções relacionadas ao botão “Tributário”
3.	O usuário seleciona a opção “Tributação”
4.	O sistema exibe a tela que contém todas as tributações cadastradas
5.	O usuário seleciona o botão “Novo”
6.	O sistema exibe a tela do formulário de cadastro de tributação contendo os seguintes campos a serem preenchidos: - Descrição - ICMS Contribuinte - ICMS Não Contribuinte - IPI - PIS - PIS ST - COFINS - COFINS ST
7.	O usuário faz o seguinte: - informa a **Descrição** “Simples Nacional” - informa a Origem CST “0”, “1”, “2” ou “3” do **ICMS Contribuinte** - informa a Situação Tributária “101”, “102”, “103”, “300”, “400” ou “500” do **ICMS Contribuinte**

	- informa a Origem CST “0”, “1”, “2” ou “3” do **ICMS Não Contribuinte** - informa a Situação Tributária “101”, “102”, “103”, “300”, “400” ou “500” do **ICMS Não Contribuinte** - informa a Situação Tributária “00”, “49”, “50” ou “99” do **IPI** - informa a Base de Cálculo “0,20” do **IPI** - informa a Alíquota “0,35” do **IPI** - informa a Situação Tributária “01”, “02” ou “99” do **PIS** - informa a Base de Cálculo “0,70” do **PIS** - informa a Alíquota “0,80” do **PIS** - informa a Base de Cálculo “0,39” do **PIS ST** - informa a Alíquota “0,78” do **PIS ST** - informa a Situação Tributária “01”, “02”, “99” da **COFINS** - informa a Base de Cálculo “0,40” da **COFINS** - informa a Alíquota “0,50” da **COFINS** - informa a Base de Cálculo “0,97” da **COFINS ST** - informa a Alíquota “0,85” da **COFINS ST** - clica no botão “Salvar”
8.	O sistema armazena os dados e exibe a mensagem “Registro salvo com sucesso!”

1.4 Fluxo Secundário A

Passos	**Ações**
7.A.1	O usuário faz o seguinte: - informa a **Descrição** “Simples Nacional” - informa a Origem CST “0”, “1”, “2” ou “3” do **ICMS Contribuinte** - informa a Situação Tributária “201”, “202”, “203” ou “900” do **ICMS Contribuinte** - informa a Modalidade “Lista Negativa (Valor)” do **ICMS ST Contribuinte** - informa a Redução “0,0050” do **ICMS ST Contribuinte** - informa a Alíquota “0,20” do **ICMS ST Contribuinte** - informa a MVA “3,20” do **ICMS ST Contribuinte** - informa a Origem CST “0”, “1”, “2” ou “3” do **ICMS Não Contribuinte**

	- informa a Situação Tributária “201”, “202” ou “203” do **ICMS Não Contribuinte** - informa a Modalidade “Lista Negativa (Valor)” do **ICMS ST Não Contribuinte** - informa a Redução “0,0040” do **ICMS ST Não Contribuinte** - informa a Alíquota “0,15 do **ICMS ST Não Contribuinte** - informa a MVA “4,30” do **ICMS ST Não Contribuinte** - informa a Situação Tributária “01”, “02”, “04”, “05”, “51”, “52”, “54” ou “55” do **IPI** - informa a Situação Tributária “03”, “04”, “05”, “06”, “07”, “08”, “09”, “49”, “50”, “51”, “52”, “53”, “54”, “55”, “56”, “60”, “61”, “62”, “63”, “64”, “65”, “66”, “67”, “70”, “71”, “72”, “73”, “74”, “75” ou “98” do **PIS** - informa a Base de Cálculo “0,39” do **PIS ST** - informa a Alíquota “0,78” do **PIS ST** - informa a Situação Tributária “03”, “04”, “05”, “06”, “07”, “08”, “09”, “49”, “50”, “51”, “52”, “53”, “54”, “55”, “56”, “60”, “61”, “62”, “63”, “64”, “65”, “66”, “67”, “70”, “71”, “72”, “73”, “74”, “75” ou “98” da **COFINS** - informa a Base de Cálculo “0,97” da **COFINS ST** - informa a Alíquota “0,85” da **COFINS ST** - clica no botão “Salvar”
7.A.2	O sistema armazena os dados e exibe a mensagem “Registro salvo com sucesso!”

1.5 Fluxo Secundário B

7.B.1	O usuário faz o seguinte: - informa a **Descrição** “Simples Nacional” - informa a Origem CST “0”, “1”, “2” ou “3” do **ICMS Contribuinte** - informa a Situação Tributária “201”, “202”, “203” ou “900” do **ICMS Contribuinte** - informa a Modalidade “Lista Negativa (Valor)” do **ICMS ST Contribuinte** - informa a Redução “0,0050” do **ICMS ST Contribuinte** - informa a Alíquota “0,20” do **ICMS ST Contribuinte** - informa a MVA “3,20” do **ICMS ST Contribuinte**

	- informa a Origem CST “0”, “1”, “2” ou “3” do **ICMS Não Contribuinte** - informa a Situação Tributária “900” do **ICMS Não Contribuinte** - informa a Modalidade “Lista Negativa (Valor)” do **ICMS Não Contribuinte** - informa a Redução “0,0040” do **ICMS Não Contribuinte** - informa a Alíquota “0,15 do **ICMS Não Contribuinte** - informa a Modalidade “Lista Negativa (Valor)” do **ICMS ST Não Contribuinte** - informa a Redução “0,0040” do **ICMS ST Não Contribuinte** - informa a Alíquota “0,15 do **ICMS ST Não Contribuinte** - informa a MVA “4,30” do **ICMS ST Não Contribuinte** - informa a Situação Tributária “01”, “02”, “04”, “05”, “51”, “52”, “54” ou “55” do **IPI** - informa a Situação Tributária “03”, “04”, “05”, “06”, “07”, “08”, “09”, “49”, “50”, “51”, “52”, “53”, “54”, “55”, “56”, “60”, “61”, “62”, “63”, “64”, “65”, “66”, “67”, “70”, “71”, “72”, “73”, “74”, “75” ou “98” do **PIS** - informa a Base de Cálculo “0,39” do **PIS ST** - informa a Alíquota “0,78” do **PIS ST** - informa a Situação Tributária “03”, “04”, “05”, “06”, “07”, “08”, “09”, “49”, “50”, “51”, “52”, “53”, “54”, “55”, “56”, “60”, “61”, “62”, “63”, “64”, “65”, “66”, “67”, “70”, “71”, “72”, “73”, “74”, “75” ou “98” da **COFINS** - informa a Base de Cálculo “0,97” da **COFINS ST** - informa a Alíquota “0,85” da **COFINS ST** - clica no botão “Salvar”
7.B.2	O sistema armazena os dados e exibe a mensagem “Registro salvo com sucesso!”

B Casos de Teste Utilizados no Estudo de Caso

A seguir é apresentado os casos de teste utilizados no estudo de caso pelos engenheiros de software.

Legenda:	Cadastro de Tributação	Edição de Tributação			Legenda:	Ok
	Filtro de Tributação	Exclusão de Tributação				Problema
						Estranho

Caso de Teste						
ID	**Título**	**Grupo**	**Descrição**			
1	Cadastrar Tributação tendo uma Descrição com o número máximo de caracteres	Tributação - Cadastro de Tributação	**Cenário: Cadastrar Tributação tendo uma Descrição com o número máximo de caracteres** Quando o usuário informa que deseja cadastrar uma nova Tributação E seleciona o botão "Novo" na tela de Cadastro de Tributações E informa a Descrição "Simples Nacionalaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa" E informa a Origem CST "0" do ICMS Contribuinte E informa a Situação Tributária "101" do ICMS Contribuinte E informa a Origem CST "0" do ICMS Não Contribuinte E informa a Situação Tributária "101" do ICMS Não Contribuinte E informa a Situação Tributária "01" do IPI E informa a Situação Tributária "03" do PIS E informa a Base de Cálculo "2,50" do PIS ST E informa a Alíquota "5,00" do PIS ST E informa a Situação Tributária "75" do COFINS E informa a Base de Cálculo "0,54" do COFINS ST E informa a Alíquota "0,51" do PIS ST E o usuário pressiona o botão "Salvar" Então o sistema exibe a mensagem "Registro salvo com sucesso!"			
2	Salvar tributação sem informar a Base de Cálculo do PIS ST	Tributação - Cadastro de Tributação	**Cenário: Salvar tributação sem informar a Base de Cálculo do PIS ST** Quando o usuário informa que deseja cadastrar uma nova Tributação E seleciona o botão "Novo" na tela de Cadastro de Tributações E informa a Descrição "Simples Nacional" E informa a Origem CST "0" do ICMS Contribuinte E informa a Situação Tributária "101" do ICMS Contribuinte E informa a Origem CST "0" do ICMS Não Contribuinte E informa a Situação Tributária "101" do ICMS Não Contribuinte E informa a Situação Tributária "01" do IPI E informa a Situação Tributária "03" do PIS E informa a Base de Cálculo "0,00" do PIS ST E informa a Alíquota "5,00" do PIS ST E informa a Situação Tributária "75" do COFINS E informa a Base de Cálculo "0,54" do COFINS ST E informa a Alíquota "0,51" do PIS ST E o usuário pressiona o botão "Salvar" Então o sistema exibe a mensagem "Registro salvo com sucesso!"	O campo tem um "*" dando a entender que é obrigatório		

3	Salvar tributação sem informar a Alíquota do PIS ST	Tributação - Cadastro de Tributação	**Cenário: Salvar tributação sem informar a Alíquota do PIS ST** Quando o usuário informa que deseja cadastrar uma nova Tributação E seleciona o botão "Novo" na tela de Cadastro de Tributações E informa a Descrição "Simples Nacional" E informa a Origem CST "0" do ICMS Contribuinte E informa a Situação Tributária "101" do ICMS Contribuinte E informa a Origem CST "0" do ICMS Não Contribuinte E informa a Situação Tributária "101" do ICMS Não Contribuinte E informa a Situação Tributária "01" do IPI E informa a Situação Tributária "03" do PIS E informa a Base de Cálculo "0,50" do PIS ST E informa a Alíquota "0,00" do PIS ST E informa a Situação Tributária "75" do COFINS E informa a Base de Cálculo "0,54" do COFINS ST E informa a Alíquota "0,51" do PIS ST E o usuário pressiona o botão "Salvar" Então o sistema exibe a mensagem "Registro salvo com sucesso!"	O campo tem um "*" dando a entender que é obrigatório		
4	Salvar tributação sem informar a Base de Cálculo do COFINS ST	Tributação - Cadastro de Tributação	**Cenário: Salvar tributação sem informar a Base de Cálculo do COFINS ST** Quando o usuário informa que deseja cadastrar uma nova Tributação E seleciona o botão "Novo" na tela de Cadastro de Tributações E informa a Descrição "Simples Nacional" E informa a Origem CST "0" do ICMS Contribuinte E informa a Situação Tributária "101" do ICMS Contribuinte E informa a Origem CST "0" do ICMS Não Contribuinte E informa a Situação Tributária "101" do ICMS Não Contribuinte E informa a Situação Tributária "01" do IPI E informa a Situação Tributária "03" do PIS E informa a Base de Cálculo "0,50" do PIS ST E informa a Alíquota "5,00" do PIS ST E informa a Situação Tributária "75" do COFINS E informa a Base de Cálculo "0,00" do COFINS ST E informa a Alíquota "0,51" do PIS ST E o usuário pressiona o botão "Salvar" Então o sistema exibe a mensagem "Registro salvo com sucesso!"	O campo tem um "*" dando a entender que é obrigatório		
5	Salvar tributação sem informar a Alíquota do COFINS ST	Tributação - Cadastro de Tributação	**Cenário: Salvar tributação sem informar a Alíquota do COFINS ST** Quando o usuário informa que deseja cadastrar uma nova Tributação E seleciona o botão "Novo" na tela de Cadastro de Tributações E informa a Descrição "Simples Nacional" E informa a Origem CST "0" do ICMS Contribuinte E informa a Situação Tributária "101" do ICMS Contribuinte E informa a Origem CST "0" do ICMS Não Contribuinte E informa a Situação Tributária "101" do ICMS Não Contribuinte E informa a Situação Tributária "01" do IPI E informa a Situação Tributária "03" do PIS E informa a Base de Cálculo "0,50" do PIS ST E informa a Alíquota "5,00" do PIS ST E informa a Situação Tributária "75" do COFINS E informa a Base de Cálculo "0,51" do COFINS ST E informa a Alíquota "0,00" do PIS ST E o usuário pressiona o botão "Salvar" Então o sistema exibe a mensagem "Registro salvo com sucesso!"	O campo tem um "*" dando a entender que é obrigatório		

6	Editar a Situação Triutária do ICSM Não Contribuinte	Tributação - Edição de Tributação	**Cenário: Editar a Situação Triutária do ICSM Não Contribuinte** Quando o usuário seleciona a tributação "Simples Nacional 1" E clica no botão "Editar" E altera a Situação Tributária para "201" do ICMS Não Contribuinte E informa a Modalidade "Lista Negativa (Valor)" do ICMS ST Não Contribuinte E informa a Redução "0,0000" ICMS ST Não Contribuinte E informa a Alíquota "0,00" do ICMS ST Não Contribuinte E informa a MVA "0,00" do ICMS ST Não Contribuinte E o usuário pressiona o botão "Salvar" Então o sistema exibe a mensagem "Registro salvo com sucesso!"	Os campos tem um "*" dando a entender que é obrigatório		
7	Editar a Base de Cálculo e Alíquota do IPI	Tributação - Edição de Tributação	**Cenário: Editar a Base de Cálculo e Alíquota do IPI** Quando o usuário seleciona a tributação "Simples Nacional 1" E clica no botão "Editar" E altera a Base de Cálculo para "0,00" do IPI E altera a Alíquota para "0,00" do IPI E o usuário pressiona o botão "Salvar" Então o sistema exibe a mensagem "Registro salvo com sucesso!"	Os campos tem um "*" dando a entender que é obrigatório		
8	Tentar excluir uma tributação e não confirmar	Tributação - Exclusão de Tributação	**Cenário: Tentar excluir uma tributação e não confirmar** Quando o usuário seleciona a tributação "Simples Nacional 1" E clica no botão "Excluir" E o sistema exibe a mensagem "Deseja realmente excluir o item: 10 - Simples Nacional 1" E clica no botão "Não" Então o sistema não exclui o item	A mensagem não tem ponto de interrogação		
9	Tentar excluir uma tributação e confirmar	Tributação - Exclusão de Tributação	**Cenário: Tentar excluir uma tributação e confirmar** Quando o usuário seleciona a tributação "Simples Nacional 1" E clica no botão "Excluir" E o sistema exibe a mensagem "Deseja realmente excluir o item: 10 - Simples Nacional 1" E clica no botão "Sim" Então o sistema exibe mensagem "Registro deletado com Sucesso!"			
10	Digitar uma descrição no campo de Pesquisa e apertar a tecla Enter	Tributação - Filtro de Tributação	**Cenário: Digitar uma descrição no campo de Pesquisa e apertar a tecla Enter** Quando o usuário digita "sim" no campo de Pesquisa E aperta a tecla Enter Então o sistema abre o formulário de uma tributação aleatória	Deveria ter o mesmo comportamento do botão "Buscar"		
11	Pesquisar por um caracter especial	Tributação - Filtro de Tributação	**Cenário: Pesquisar por um caracter especial** Quando o usuário digita "%" no campo de Pesquisa E clica no botão "Buscar" Então o sistema não filtra o item que tem a "%" na descrição			
12	Filtrar pela descrição	Tributação - Filtro de Tributação	**Cenário: Filtrar pela descrição** Quando o usuário digita "simples" no campo de Pesquisa E clica no botão "Buscar" Então o sistema retorna o item que tem o código "simples"			
13	Filtrar pelo código	Tributação - Filtro de Tributação	**Cenário: Filtrar pelo código** Quando o usuário digita "5" no campo de Pesquisa E clica no botão "Buscar" Então o sistema retorna o item que tem o código "5"	OBS: os códigos não estão em ordem		

14	Filtrar palavras sem acento gráfico	Tributação - Filtro de Tributação	**Cenário: Filtrar palavras sem acento gráfico** Quando o usuário digita "comercio" no campo de Pesquisa E clica no botão "Buscar" Então o sistema retorna a mensagem "nenhum item encontrado"	Deveria retornar o item que tem a palavra "comércio" na descrição		

Referências

AJILA, S. A.; KABA, A. B. Using traceability mechanisms to support software product line evolution. In: IEEE. *Information Reuse and Integration, 2004. IRI 2004. Proceedings of the 2004 IEEE International Conference on*. [S.l.], 2004. p. 157–162. Citado na página 34.

BUBAK, O.; GOMAA, H. Applying software product line concepts in service orientation. *International Journal of Intelligent Information and Database Systems*, Inderscience Publishers, v. 2, n. 4, p. 383–396, 2008. Citado na página 28.

CALERO, J. M. A. et al. Toward a multi-tenancy authorization system for cloud services. *IEEE Security & Privacy*, IEEE, v. 8, n. 6, p. 48–55, 2010. Citado na página 19.

CAPILLA, R.; DUEÑAS, J. C. Evolution and maintenance of web sites: A product line model. *TeAM YYePG*, p. 255, 2005. Citado na página 33.

CAPILLA, R.; TOPALOGLU, N. Y. Product lines for supporting the composition and evolution of service oriented applications. In: IEEE. *Principles of Software Evolution, Eighth International Workshop on*. [S.l.], 2005. p. 53–56. Citado 2 vezes nas páginas 19 e 32.

CLEMENTS, P.; NORTHROP, L. *Software product lines: practices and patterns*. [S.l.]: Addison-Wesley, 2002. Citado na página 21.

COHEN, S. *Product line state of the practice report*. [S.l.], 2002. Citado 2 vezes nas páginas 17 e 21.

COHEN, S. *Predicting when product line investment pays*. [S.l.], 2003. Citado 2 vezes nas páginas 21 e 22.

CRAMON, J. *Microservices: Usage Is More Important than Size*. 2017. Disponível em? https://www.infoq.com/news/2014/05/microservices-usage-size/. Acesso em: 2018-01-29. Citado na página 27.

DRAGONI, N. et al. Microservices: How to make your application scale. *arXiv preprint arXiv:1702.07149*, 2017. Citado na página 26.

DURSCKI, R. C. et al. Linhas de produto de software: riscos e vantagens de sua implantação. *VI Simpósio Internacional de Melhoria de Processos de Software*, p. 155–166, 2004. Citado na página 22.

FUGGETTA, A. et al. Applying gqm in an industrial software factory. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, ACM, v. 7, n. 4, p. 411–448, 1998. Citado na página 47.

HORCAS, J.-M.; PINTO, M.; FUENTES, L. Product line architecture for automatic evolution of multi-tenant applications. In: IEEE. *Enterprise Distributed Object Computing Conference (EDOC), 2016 IEEE 20th International*. [S.l.], 2016. p. 1–10. Citado 2 vezes nas páginas 19 e 33.

KANG, K. C. et al. Form: A feature-; oriented reuse method with domain-; specific reference architectures. *Annals of Software Engineering*, Springer, v. 5, n. 1, p. 143, 1998. Citado na página 30.

LEWIS, J.; FOWLER, M. Microservices: a definition of this new architectural term. *Mars*, 2014. Citado 13 vezes nas páginas 18, 19, 20, 24, 27, 30, 37, 42, 43, 45, 59, 61 e 62.

LINDEN, F. vd; SCHMID, K.; ROMMES, E. *Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering. Secaucus*. [S.l.]: NJ, USA: Springer-Verlag New York, Inc, 2007. Citado na página 39.

MOREIRA, P. F. M.; BEDER, D. M. Desenvolvimento de aplicações e micro serviços: Um estudo de caso. *Revista TIS*, v. 4, n. 3, 2016. Citado 2 vezes nas páginas 18 e 25.

NAILY, M. A. et al. A framework for modelling variable microservices as software product lines. In: SPRINGER. *International Conference on Software Engineering and Formal Methods*. [S.l.], 2017. p. 246–261. Citado 2 vezes nas páginas 19 e 31.

NEWMAN, S. *Building microservices: designing fine-grained systems*. [S.l.]: "O'Reilly Media, Inc.", 2015. Citado 12 vezes nas páginas 18, 19, 20, 24, 27, 37, 42, 43, 45, 59, 61 e 62.

O'GRADY, S. *The Difference Between SOA and Microservices Isn't Size*. 2017. Disponível em: http://redmonk.com/sograady/2017/07/20/soa-microservices/. Acesso em: 2018-01-29. Citado na página 27.

PEREIRA, J. A.; FIGUEIREDO, E.; COSTA, H. Linha de produtos de software: Conceitos e ferramentas. 2015. Citado na página 23.

PRESSMAN, R. S. *Software engineering: a practitioner's approach*. [S.l.]: Palgrave Macmillan, 2005. Citado 2 vezes nas páginas 49 e 62.

SCHUBANZ, M. et al. Modeling rationale over time to support product line evolution planning. In: ACM. *Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems*. [S.l.], 2012. p. 193–199. Citado na página 36.

SCHUBANZ, M. et al. Model-driven planning and monitoring of long-term software product line evolution. In: ACM. *Proceedings of the Seventh International Workshop on Variability Modelling of Software-intensive Systems*. [S.l.], 2013. p. 18. Citado na página 36.

STOIBER, R.; GLINZ, M. Feature unweaving: Efficient variability extraction and specification for emerging software product lines. In: IEEE. *2010 Fourth International Workshop on Software Product Management*. [S.l.], 2010. p. 53–62. Citado na página 17.

TEIXEIRA, L.; BORBA, P.; GHEYI, R. Safe evolution of product populations and multi product lines. In: ACM. *Proceedings of the 19th International Conference on Software Product Line*. [S.l.], 2015. p. 171–175. Citado na página 36.

THÖNES, J. Microservices. *IEEE Software*, IEEE, v. 32, n. 1, p. 116–116, 2015. Citado na página 18.

TIZZEI, L. P. et al. Using microservices and software product line engineering to support reuse of evolving multi-tenant saas. In: ACM. *Proceedings of the 21st International Systems and Software Product Line Conference-Volume A*. [S.l.], 2017. p. 205–214. Citado 9 vezes nas páginas 19, 29, 30, 37, 42, 50, 56, 60 e 61.

WOHLIN, C. et al. *Experimentation in software engineering*. [S.l.]: Springer Science & Business Media, 2012. Citado 4 vezes nas páginas 20, 46, 48 e 60.