



Universidade Federal de Pernambuco
Centro de Informática
*Graduação em Sistemas de Informação*

**Dênio Batista Brasileiro Bezerra**

# Análise do Impacto da Estratégia de Uso de Branching em Devops

Recife, Dezembro de 2017



Universidade Federal de Pernambuco
Centro de Informática
*Graduação em Sistemas de Informação*

**Dênio Batista Brasileiro Bezerra**

# Análise do Impacto da Estratégia do Uso de Branching em Devops

Projeto de Graduação apresentado no Centro de Informática da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do grau de Bacharel em Sistemas de Informação.

Orientador: Vinícius Cardoso Garcia

Co-orientador: José Fernando Carvalho

Recife, Dezembro de 2017

# FOLHA DE APROVAÇÃO

ANÁLISE DO IMPACTO DA ESTRATÉGIA DE USO DE BRANCHING EM DEVOPS

Projeto de Graduação apresentado no Centro de Informática da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do grau de Bacharel em Sistemas de Informação.

Orientador: Vinícius Cardoso Garcia

Co-orientador: José Fernando Carvalho

Data da aprovação: / / .

Banca Examinadora:

_____

Prof. Dr. Vinícius Cardoso Garcia

_____

Prof. Dr. Kiev Santos da Gama

Recife
2017

# AGRADECIMENTOS

Primeiramente, gostaria de agradecer a Deus por esta conquista, pois sem Ele seria impossível eu chegar até aqui, seu amor e a sua misericórdia me acompanharam durante todo este trajeto repleto de momentos difíceis, me dando a rocha necessária para eu conseguir alcançar este objetivo.

Também gostaria de agradecer a minha família, meus pais, Dênio e Bete, por sempre terem acreditado no meu potencial, e pelo esforço enorme para me dar uma educação de qualidade, tratando a mesma como prioridade durante toda a minha vida, e pelo amor incondicional que tem por mim. Gostaria de agradecer também a minha irmã Elisa, que sempre esteve na torcida por mim, me apoiando e incentivando a alcançar objetivos maiores.

Em especial, agradeço a minha namorada, Vanessa, uma verdadeira companheira, através de seu amor e carinho sem tamanho, foi fundamental durante este trajeto, estando ao meu lado nos momentos mais difíceis da graduação, me apoiando e incentivando a lutar por nossa futura família.

Ao meu orientador, Vinícius Garcia, pelos ensinamentos neste trabalho e durante toda a graduação, se tornando mais do que um professor, mas sim um amigo.

Ao meu co-orientador, José Fernando Carvalho, pelo seu esforço e atenção, me passando ensinamentos fundamentais para que eu concluísse este trabalho.

Aos meus amigos, Bruno, Wanderson e Marcílio, que venceram comigo muitos obstáculos encontrados neste trajeto, e que vou levar para sempre em minha vida.

Por fim agradeço ao Centro de Informática (Cin-UFPE) por me fornecer uma estrutura de excelência para o aprendizado, e a todos os professores, pois me passaram ensinamentos fundamentais para o meu sucesso acadêmico e profissional.

# RESUMO

Atualmente, estamos vivendo uma transformação digital onde as pessoas estão cada vez mais ligadas a tecnologia, buscando cada vez mais softwares de qualidade, paralelamente, a indústria tecnológica também está se transformando do ponto de vista cultural, com práticas e ferramentas sendo inseridas para aumentar a qualidade dos softwares produzidos. Fazendo parte desse avanço, surgiu em meados de 2009, o movimento *DevOps*, que consiste em um conjunto de práticas focadas em um trabalho mais colaborativo entre os times de desenvolvimento e de operações, e tem na integração contínua uma parte essencial para ser implementado. Por outro lado, a técnica de branching é cada vez mais utilizada pelos indivíduos deste tipo de indústria, e isso se deve ao fato de que este tipo de técnica possibilita o de desenvolvimento em paralelo por várias pessoas, porém o uso dessa técnica deve possuir cuidados para evitar a incidência de anti-padrões. Este trabalho foi desenvolvido através de uma pesquisa exploratória para averiguar a incidência de anti-padrões de branching em organizações que possuem a cultura DevOps.

**Palavras chave:** DevOps, Branching, Integração Contínua, Desenvolvimento de Software, Software Reliability Engineering, Gerência de Configuração de Software. .

# ABSTRACT

Currently, we are living a digital transformation where people are increasingly linked to technology, seeking more and more quality software, in parallel, the technology industry is also transforming from a cultural point of view, with practices and tools being inserted to increase the quality of the software produced. As part of this breakthrough, the DevOps movement began in mid-2009, consisting of a set of practices focused on a more collaborative work between the development and operations teams, and has in continuous integration an essential part to be implemented. On the other hand, the technique of branching is increasingly used by individuals of this type of industry, and this is due to the fact that this type of technique allows the development of parallel development by several people, but the use of this technique must have care to avoid the incidence of anti-patterns. This work was developed through an exploratory research to investigate the incidence of anti-branching patterns in organizations that have the DevOps culture.

**Keywords**: DevOps, Branching, Continuous Integration, Software Development, Software Reliability Engineering, Software Configuration Management. .

# SUMÁRIO

# Lista de Figuras

# Lista de Gráficos

# Lista de Abreviações

**GCS:** Gestão de Configuração de Software

**SW:** Software

**SaaS:** Software como Serviço

**ES:** Engenharia de Software

**IC:** Integração Contínua

**TI:** Tecnologia da Informação

# 1.Introdução

Este capítulo apresentará o contexto e a motivação para a realização deste trabalho, bem como os objetivos, geral e específicos, a serem atingidos a fim de auxiliar um bom entendimento ao leitor.

## 1.1 Contexto

Com o avanço da tecnologia, o uso de software se tornou essencial para a sociedade, o surgimento dos computadores e a popularização da internet, tornaram seu uso, uma ação diária na vida humana. Segundo Pressman [1] o software está profundamente incorporado em praticamente todos os aspectos da nossa vida. Isso fica mais evidente ao repararmos como nossos hábitos básicos mudaram, como trabalhar, estudar, se comunicar e até mesmo de onde gastar nosso tempo livre, dessa forma, os usuários e os desenvolvedores estão olhando cada vez com mais atenção para a qualidade dos softwares hoje produzidos, atributos como facilidade de manutenção, confiança, eficiência e usabilidade são cada vez mais exigidos [3], por esse motivo é importante investir na qualidade do software desde o momento em que a primeira linha de código é escrita até o momento em que ele está em produção [2].

É um erro a associação do termo software somente aos programas de computadores, uma vez que software não é apenas o programa, mas também toda a documentação associada e os dados de configuração necessários para fazer com que esses programas operem corretamente [3]. Além do mais a grande maioria dos softwares hoje em dia não operam sozinhos, com o surgimento dos SaaS (Software como Serviço) e da computação em nuvem os sistemas devem ter uma grande facilidade em se comunicar com outros tipos de sistemas, portanto quando um

software não tem seu funcionamento em um bom nível, ele não atinge somente a si, mas a todos aqueles softwares que depende dele para realizar algum tipo de processo próprio.

Para o software conseguir atingir um bom nível de qualidade existem várias técnicas e ferramentas, que se desenvolveram muito nos últimos anos, e que abrangem todo o ciclo de vida do software. Uma delas é a Gerência de Configuração de Software (GCS), que é definida como a disciplina de identificar a configuração de um sistema em diferentes pontos no tempo com a finalidade de controlar sistematicamente as mudanças realizadas, mantendo a integridade e rastreabilidade da configuração através do ciclo de vida do sistema [21]. Uma segunda definição diz que a GCS consiste num conjunto de atividades projetadas para controlar as mudanças através da identificação de artefatos, estabelecendo um relacionamento entre eles, definindo o mecanismo para o gerenciamento de diferentes versões destes produtos, controlando as mudanças impostas, auditando e relatando as mudanças realizadas [2].

Essa necessidade se faz presente por vários motivos, um deles é o fator humano. Segundo Babich em toda equipe há um certo grau de desordem, independente do tamanho da equipe e dos indivíduos que a compõem. Outro motivo ainda mais crítico, segundo a literatura, é o fator “mudança”, pois ao longo do ciclo de vida de um software, diversas modificações podem ocorrer em seu projeto original, os motivos e origens destas modificações podem ser os mais variados possíveis e podem ocorrer em qualquer época [1].

Este tipo de problema causa uma grande preocupação para os engenheiros de SW, sendo abordado no primeiro mandamento da engenharia definido por Meira et. al. [22], que foi criado com o propósito de criar um conjunto universal de leis de software a partir da junção de algumas das mesmas, para que assim seja formada uma cultura universal única a respeito da engenharia de SW. Meira et. al., definiu em uma das partes do mandamento que um SW deve verificar as condições do seu ambiente operacional e decidir se suas ações são coerentes ou não com o contexto

atual desse ambiente. Além disso, o próprio sistema, deve poder realizar a mudança em si próprio, pedir uma modificação externa ou ser modificado por algum tipo de agente externo [22], porém, nos dias atuais a grande maioria dos sistemas não funcionam dessa forma, causando grandes problemas para as organizações que as utilizam e para os seus desenvolvedores.

Com o surgimento do manifesto ágil [4] os problemas relacionados às mudanças nos SW se tornaram ainda mais presentes, tornando-se uma parte crítica deste tipo de desenvolvimento de SW, pois ao adotarmos esse modelo nos inserimos em um ambiente onde ocorrem mudanças e implementações de risco em um espaço curto de tempo, com os desenvolvedores atuando no mesmo repositório e ao mesmo tempo gerando uma grande quantidade de informações que devem ser integradas, consequentemente podemos ter o aumento do número de erros no seu desenvolvimento ou na sua operação, causando bastante dano para a indústria, que possui, atualmente, o desenvolvimento ágil como padrão.

Grande parte desses problemas tem como causa a falta de trabalho colaborativo entre o time de desenvolvimento e o time de operações, pois os dois times possuem por cultura, pensamentos distintos sobre suas responsabilidades, enquanto o time de desenvolvimento foca em criar novos produtos e aplicações, adicionar funcionalidades ou corrigir bugs, o time de operações fica responsável por cuidar desses produtos e aplicações em produção [6], o time de desenvolvimento busca basicamente introduzir funcionalidades e mudanças da melhor forma, enquanto o time de operações busca pela estabilidade do sistema [2], essa falta de colaboração entre os dois times acaba aumentando o risco de se possuir códigos defeituosos na aplicação.

Para conseguir fazer a integração entre estes dois times, foi criado em meados de 2009 [5], o movimento DEVOPS, que consiste em um conjunto de práticas destinadas a reduzir o tempo entre armazenar uma mudança (nesse caso uma nova porção de código desenvolvida) em um sistema de controle de versões e sua execução em ambiente de produção, garantindo alta qualidade [5] e que visa

melhorar a comunicação entre desenvolvedores e o time de operações para resolver problemas críticos, como o medo da mudança e as implantações de risco [6].

.

Entre as técnicas usadas para a gestão de configuração no desenvolvimento de software, temos a técnica do uso de branching, que é usado para o trabalho simultâneo em times de desenvolvimento, onde os desenvolvedores trabalham através de ramos(branch) separados e em paralelo, conseguindo assim a garantia de que não vão causar impacto na linha principal do código, obtendo a estabilidade necessária para o software [8], e introduzir esse tipo de técnica em um ambiente DevOps pode trazer benefícios ou não, por esse motivo essa pesquisa visa realizar um estudo sobre o impacto da estratégia de branching em ambientes DevOps, essa pesquisa será baseada nos anti-padrões de branching definidas por Appleton et. al. [28], onde será avaliado qual o grau de impacto que esses anti-padrões tem em organizações que possuem a cultura DevOps. .

## 1.2 Motivação

A transformação digital que estamos vivendo nos trouxe avanços nas práticas utilizadas no desenvolvimento de software. Isso se dá principalmente ao fato das empresas estarem buscando qualidade não só da sua aplicação final, mas também da sua produtividade durante a operação. Para assegurar essa qualidade durante todo o ciclo de vida do software, é de suma importância a prática da gestão de configuração de software, pois consiste em práticas e ferramentas para fazer o gerenciamento dos mais diversos tipos de artefatos que são gerados durante o processo de desenvolvimento até a sua operação. Com o passar dos anos surgiram várias técnicas e ferramentas para serem utilizadas nesse tipo de prática, porém

sua aplicação se torna crítica quando não adotamos a forma correta de aplicá-la [14].

Na GCS o artefato que requer maior atenção é o código fonte, por ter muitas vezes várias pessoas trabalhando no mesmo código e por ser um ambiente muito propício a mudanças, se torna um processo de muitas incertezas, e por isso devemos escolher bem a técnica a ser utilizada durante esse processo.

O surgimento do Devops nos trouxe a necessidade de investigar os tipos de técnicas existentes e sua aplicação no mesmo, pois tal movimento nos trouxe possibilidades de velocidade e automação para o desenvolvimento de software, e uma técnica mal escolhida pode acarretar em problemas críticos para toda a equipe conseguir alcançar resultados satisfatórios e a qualidade pretendida.

Dentre as técnicas existentes optamos por estudar a técnica de branching, por ser bastante utilizada nas práticas convencionais para trabalhos simultâneos entre times de desenvolvimento, iremos investigar o grau de incidência dos anti-padrões de branching definidos por appleton et al., em organizações que possuem a cultura DevOps.

## 1.3 Objetivos

### 1.3.1 Geral

Este trabalho de graduação tem como objetivo compreender e analisar conceitos e práticas pertinentes ao uso da técnica de branching em Devops, especialmente sobre o impacto causado pelos anti-padrões desta técnica definidos por Appleton et. al. no processo de desenvolvimento de software nas empresas que utilizam esta prática. O estudo se dará através de uma pesquisa quantitativa a ser realizada com o intuito de comprovar ou refutar as hipóteses [26].

### 1.3.2 Específicos

- Apresentar conceitos encontrados na literatura existente sobre DevOps e Branching

- Averiguar a incidência dos anti-padrões de branching no desenvolvimento de software
- Identificar os relacionamentos existentes entre a técnica e seus anti-padrões e a cultura DevOps

- Investigar e analisar o impacto da técnica de branching na cultura DevOps

## 1.4 Organização do trabalho

O presente trabalho está organizado em 5 capítulos, dos quais o primeiro trata-se da introdução e os 4 demais podem ser descritos abaixo:

- Capítulo 2: apresenta um conjunto de definições sobre Engenharia de Software, Desenvolvimento de Software, Desenvolvimento Ágil,Gerência de Configuração de Software, DevOps, Integração Contínua e Entrega Contínua.
- Capítulo 3: Análise do impacto de Branching em Devops
- Capítulo 4: São apresentadas considerações finais do trabalho assim como as propostas para trabalhos futuros.

# 2. Fundamentação Teórica

Este capítulo abordará conhecimentos essenciais para a boa compreensão e continuidade deste trabalho. Propondo uma visão geral sobre os temas: Engenharia de Software, Gerência de Configuração de Software, DevOps, Integração Contínua e Branching.

## 2.1 Desenvolvimento de Software

Para discutir os problemas relacionados ao desenvolvimento de software, foi criado em 1969, o termo “engenharia de software”, a fim de resolver alguns problemas que eram enfrentados na época como o atraso na entrega de grandes softwares, que além disso, não entregavam a funcionalidade de que os usuários necessitavam, custavam mais do que o esperado e não eram confiáveis [3]. Fritz Bauer [1] definiu o conceito de ES como "Engenharia de Software é o estabelecimento e o emprego de sólidos princípios de engenharia de modo a obter software de maneira econômica, que seja confiável e funcione de forma eficiente em máquinas reais" [1]. Com uma definição ainda mais clara a IEEE definiu ES como "A aplicação de uma abordagem sistemática, disciplinada e quantificável no desenvolvimento, na operação e na manutenção de software; isto é, a aplicação de engenharia de software" [1].

Isso se aplica diretamente no produto, no caso o software em desenvolvimento, Segundo Sommerville, software é caracterizado como um

programa de computador e toda a documentação associada a ele [3]. Já Pressman define software como um elemento de sistema lógico, e não físico que não se desgasta [1]. Por ser uma área bastante extensa e com várias formar de ser aplicada notou-se a necessidade de desenvolvê-la ao passar dos anos, com isso a forma de se construir software foi mudando, nasceram técnicas e ferramentas específicas para auxiliar este processo.

## 2.2 Processos de Desenvolvimento de Software

Uma das principais atividades da engenharia de software é o desenvolvimento de software, que segundo a literatura é definido também como um processo de software, um processo de software é uma sequência de atividades que leva à produção de um produto de software [3]. Existem quatro atividades fundamentais comuns a todos os processos de software. São elas:

- Especificação de software, em que clientes e engenheiros definem o software a ser produzido e as restrições de sua operação.
- Desenvolvimento de software, em que o software é projetado e programado.
- Validação de software, em que o software é verificado para garantir que é o que o cliente quer.
- Evolução de software, em que o software é modificado para refletir a mudança de requisitos do cliente e do mercado.

Existem várias metodologias de desenvolvimento, as mais tradicionais são os modelos cascata, espiral, incremental e evolucionário [1], esses modelos são geralmente baseados em entrega de funcionalidades onde são especificados, desenvolvidos e testados em estágios sequenciais e lineares [6], prezando pelo comprimento das suas fases, com uma grande quantidade de documentação sendo gerada durante seu ciclo de vida. Levando-se em conta que um processo de desenvolvimento de software pode sofrer muitas mudanças, seja ela por necessidade da equipe ou do cliente, é necessário alterar constantemente o produto e a documentação, gastando-se muito tempo para realizar tais alterações.

## 2.3 Desenvolvimento Ágil

Os modelos mais tradicionais como cascata, incrementais, espirais e evolucionários, são por base dirigidos a planos [3], trazendo assim alguns problemas como gasto de tempo excessivo com documentação, dificuldades para realizar mudanças de forma rápida, além de que um erro descoberto durante o processo de desenvolvimento se torna caro e complicado para ser solucionado [1]. Levando-se em consideração que o mundo da tecnologia é bastante mutável, com mudanças no código sendo solicitadas em espaços curtos de tempo, seja pelo cliente ou pela equipe de desenvolvimento, gastar muito tempo com mudanças acaba sendo um problema para qualquer organização de TI. Diante disso, surgiu em 2001 o manifesto ágil, com o intuito de diminuir o tempo gasto com a documentação e o planejamento, para atribuir um maior foco ao código em desenvolvimento. O manifesto surgiu com 4 valores a serem colocados em prática [4]:

- **Indivíduos e interações** mais que processos e ferramentas
- **Software em funcionamento** mais que documentação abrangente
- **Colaboração com o cliente** mais que negociação de contratos
- **Responder a mudanças** mais que seguir um plano

O desenvolvimento ágil propõe algumas mudanças em relação aos modelos tradicionais como entregar o software em funcionamento rapidamente aos clientes, e estes poderão, em seguida, propor alterações e novos requisitos a serem incluídos nas iterações posteriores do sistema [3]. A chegada do manifesto causou grande impacto na indústria de tecnologia desde o seu surgimento até os dias atuais, a metodologia ágil está cada vez mais presente nas organizações, e esse tipo de metodologia não é exclusividade da equipe de desenvolvimento, mas atualmente é também utilizado pelos gestores, se propagando para todos os tipos de organizações [24].

Esta metodologia alinha ainda mais o que está em desenvolvimento e suas mudanças ao contexto de negócio da aplicação, isso ocorre devido ao foto que no desenvolvimento ágil, vários grupos da organização trabalham em conjunto, como desenvolvedores, gestores e testadores, além de que o cliente/usuário tem participação direta sobre o que está sendo desenvolvido, porém, mesmo com essa nova metodologia, o time de operações seguiu trabalhando em separado, apenas recebendo o que era desenvolvido e testado, para colocar no ambiente de produção, sem possuir integração com os outros times.

## 2.4 Problemas Enfrentados

Com o avanço das práticas utilizadas para o desenvolvimento de software, as cobranças acerca da qualidade e da agilidade no desenvolvimento do software hoje produzido aumentaram [3]. O surgimento do DevOps causou um impacto positivo na indústria de software, pois trouxe ganhos práticos, na pesquisa realizada por Rebellabs foi constatado que o uso de DevOps reduz em 22% o tempo gasto em trabalhos não planejados ou em retrabalho, aumentando assim em 29% o tempo gasto para fazer novos códigos e implantar novas funcionalidades [24], porém, trouxe consigo várias práticas a serem introduzidas no ambiente organizacional das empresas de TI, entre elas a integração contínua, que por sua vez, é parte essencial para se conseguir aplicar o DevOps [2]. Essas mudanças deixaram o desenvolvimento ainda mais dinâmico, as práticas de desenvolvimento em paralelo ficaram mais controláveis, sendo auxiliadas pelas ferramentas de GCS. Além do mais, o uso de desenvolvimento de forma paralela, isolada e independente através de ramos, denominado como branching [14], aumentou significativamente entre as equipes, principalmente em equipes com número elevado de indivíduos.

Neste contexto, se faz necessário alinhar essas práticas para analisarmos o impacto que o uso de branching causa sobre a prática do DevOps, mais especificamente se causa impacto direto na integração contínua durante o

desenvolvimento do SW, para que assim seja definido, ou não, o modo como as práticas irão ser alinhadas e integradas

## 2.5 Gerência de Configuração de Software

Uma das práticas necessárias para se obter um software de boa qualidade é a gestão de configuração de software, por conta das constantes mudanças durante o desenvolvimento e até mesmo após estar em uso. Bugs são descobertos e precisam ser corrigidos. Os requisitos do sistema mudam e é preciso implementar essas mudanças em uma nova versão do sistema [3], para que assim o software consiga atender todas as necessidades dos stakeholders durante todo o seu ciclo de vida. Essas mudanças aumentam o nível de confusão entre os membros de uma equipe de software [1] e controlar essa confusão é um grande desafio, Babich afirma que a arte de coordenar desenvolvimento de software para minimizar a confusão é chamada de gestão de configuração. A gestão de configuração é a arte de identificar, organizar e controlar modificações no software que está sendo criado por uma equipe de programação. O objetivo é maximizar a produtividade minimizando os erros [1]. Essas mudanças podem ocorrer a qualquer momento e por qualquer razão, a Primeira lei da Engenharia de Software diz que: Não importa onde você esteja no ciclo de vida do sistema, o sistema mudará e o desejo de alterá-lo persistirá através de todo o ciclo de vida [10]. Além do mais a ausência de GCS pode tornar um caos o SW em desenvolvimento e por consequência prejudicar a qualidade do SW e aumentar os seus prazos de entrega [1].

O gerenciamento de configurações de um produto de sistema de software envolve quatro atividades [3]:

- **Gerenciamento de mudanças:** Envolve manter o acompanhamento das solicitações dos clientes e desenvolvedores por mudanças no software,

definir os custos e o impacto de fazer tais mudanças, bem como decidir se e quando as mudanças devem ser implementadas.

- **Gerenciamento de versões:** Envolve manter o acompanhamento de várias versões de componentes do sistema e assegurar que as mudanças nos componentes, realizadas por diferentes desenvolvedores, não interfiram umas nas outras.
- **Construção do sistema:** É o processo de montagem de componentes de programa, dados e bibliotecas e, em seguida, compilação e ligação destes, para criar um sistema executável.
- **Gerenciamento de releases:** Envolve a preparação de software para o release externo e manter o acompanhamento das versões de sistema que foram liberadas para uso do cliente.

### 2.5.1 Repositório

O processo de desenvolvimento de software gera um grande volume de informações a serem armazenadas, desta forma é necessário se ter uma centralização de tais informações para facilitar o gerenciamento, isso fica ainda mais evidente quando o time de desenvolvimento possui vários desenvolvedores trabalhando juntos no mesmo projeto [12]. Esse gerenciamento é realizado através de um repositório central, onde todos os dados necessários para o desenvolvimento devem estar armazenados nele, incluindo scripts de teste, arquivos de propriedades, esquema de banco de dados, scripts de instalação e bibliotecas de terceiros [7]. Além disso o uso do repositório evita retrabalho, sendo crucial para conseguir ter integração contínua, para fazer este gerenciamento é de suma importância usar um sistema de controle de versões, pois auxilia no controle das modificações dos itens e fornece o histórico de alterações que foram realizadas desde o primeiro dia do projeto até o estado atual, uma das características dos sistemas de controle de versão (SCV) é que eles permitem a criação de vários ramos a serem desenvolvidos em paralelo. SCV é composto por procedimentos e

ferramentas que permitem a administração do repositório e o versionamento de software [13], conseguindo assim facilitar  gerenciamento para realizar a Integração Contínua (IC).

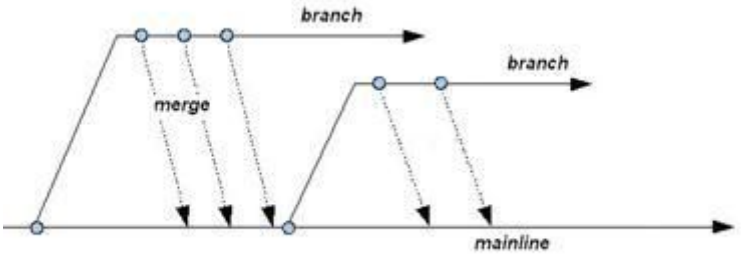
### 2.5.1.2 Git

O Git é definido como um sistema de controle de versão distribuído, que preza pela velocidade, sendo utilizado para fazer o gerenciamento de código fonte de projetos de tamanhos variados [30]. Ele foi idealizado e desenvolvido por Linus Torvalds (um dos criadores do linux) em 2005, .ao surgir a necessidade de gerenciamento do código do Kernel do linux. Seu uso cresce cada vez mais, paralelamente ao crescimento de trabalhos colaborativos em um mesmo código, pois o git facilita o gerenciamento de versões de softwares desenvolvidos colaborativamente, além de fornecer um histórico com a possibilidade de restaurar alguma versão sempre que necessário. Para todas essas possibilidades o Git conta com vários comandos, como *Commit*, que ocorre basicamente quando alguma alteração é realizada, o *Clone* que é quando o desenvolvedor copia a versão atual de algum repositório para sua máquina de trabalho local, além de *Branch* que é quando um novo ramo do projeto é criado, o *Merge* que é a operação de junção das branchs o *Fork* que é quando é realizado a cópia de um repositório para outro repositório [31].

Para realizar suas operações, o Git opera em três estados, independente do tipo de projeto, são eles, o estado *committed,* o estado *modified,* e o estado *staged.* O estado *committed* ocorre quando seus dados estão armazenados na sua base de dados local, modified é quando um arquivo sofreu alterações mas que ainda não foi consolidado na base de dados, já o staged é quando um arquivo modificado em sua versão atual para fazer parte do próximo commit [30].

### 2.5.2 Branching

A GCS também permite que a implementação de novas funcionalidades por uma equipe seja realizada pelos seus integrantes de forma paralela, isolada e independente das modificações de outros desenvolvedores. Esse isolamento é obtido com o uso de ramo (*branch*) [2]. A linha mais importante no desenvolvimento é a linha principal (mainline), também conhecida como tronco ou mestre, *Branching* significa desviar-se da linha principal e criar um novo ramo [15]. Uma *branch* é uma cópia de um certo ponto da linha principal para ser desenvolvida sem atingir o software em operação, e que após o desenvolvimento da branch é incorporada novamente a linha principal através de uma operação de junção, também conhecida como *merge* [14]. A figura 2 ilustrada por Dantas mostra este processo de se criar um branch através de algum ponto da linha principal, desenvolvê-lo e integrá-lo de volta através do merge.




Processo de branching [14]

Para facilitar o gerenciamento do processo de branching, são utilizados os softwares de controles de versão, como o git, possuindo algumas atribuições que facilitam isso, como histórico das branchs.

O surgimento de *Branching* trouxe algumas facilidades para os desenvolvedores pois o uso de ramos permite que os desenvolvedores atuem de forma isolada, isso faz com que seja reduzido os riscos de atrapalhar o trabalho de outro desenvolvedor, e nem de prejudicar o funcionamento do código principal [7], além disso aumenta o poder de recuperação do software em relação a erros graves, pois com essa técnica fica mais fácil o software voltar a um estado anterior, além disso ele permite o desenvolvimento isolado da parte de produção, diminuindo o risco de prejudicar o funcionamento do software e garantindo estabilidade para a aplicação [15].

Porém, se o uso desta técnica não for bem gerenciado existe a possibilidade da incidência de anti-padrões que podem ser definidos como situações que acontecem que prejudicam a produtividade do time que usa este tipo de técnica [28], esses anti-padrões foram definidos por Appleton et. al., e serve para ser verificado se a técnica de branching está ou não sendo bem aplicada na organização. Abaixo segue a lista de anti-padrões definidos por Appleton et. al.:

**Merge Paranóia** - É quando se tem medo de realizar o merge, geralmente devido ao medo das conseqüências.

**Merge Mania** - Gasta muito tempo na fusão do código em vez de desenvolvê-lo.

**Big Bang Merge** - É quando se tenta juntar várias branchs de uma vez só.

**Never-Ending Merge** - Uma operação de merge que nunca termina, pois sempre existe algo para ser feito o merge

**Wrong-Way Merge** - É quando o merge é realizado na branch errada.

**Branch Mania** - Branchs são criadas mais do que o necessário.

**Cascading Branches** - Os ramos são criados a vários níveis mas nunca voltam a linha principal do código.

**Mysterious Branches** - Branchs são criadas sem necessidade.

**Runaway Branches** - A finalidade da branch acaba mudando durante seu desenvolvimento e perdendo sua importância.

**Volatile Branches** - branchs são criadas com arquivos que dependem de outras branchs que não são estáveis.

**Development Freeze** - Outras atividades são paralisadas para ser feito o merge.

**Integration Wall** - Em vez de dividir o trabalho, as branchs acabam por tirar a colaboratividade do time. de desenvolvimento.

**Spaghetti Branching** - Mudanças são realizadas entre branchs que não se relacionam

### 2.5.3 Merge

Em projetos onde existe um grande número de pessoas envolvidas, é comum que elas possuam uma cópia de trabalho da linha principal de desenvolvimento. Neste contexto, o conceito de merge é também uma parte fundamental do controle de versões [16]. Merge é definido como uma operação que consiste em realizar a fusão automática das versões, gerando uma única versão que agrega todas as alterações, onde algoritmos calculam a diferença de linhas e criam uma versão final, integrando as alterações realizadas por diferentes desenvolvedores [17]. Algumas ferramentas fornecem mecanismos automatizados para realizar essa operação, pois no melhor cenário possível, ou seja, quando não ocorrem conflitos a operação pode ser feita automaticamente, porém em alguns dos casos ocorrem problemas de fusão, sendo necessária a intervenção humana para serem solucionados.

### 2.5.4 Fork

Outra técnica usada para o desenvolvimento de software é o fork, que consiste na criação de um repositório novo com tronco próprio, porém, derivado de algum outro repositório já existente [19], ela é bastante usada para criar uma bifurcação que poderá ser modificada e testada para correção de erros, enquanto uma versão se mantém estável sem ser atingida pelas mudanças, conseguindo assim manter a disponibilidade da aplicação.

### 2.5.5 Release

Dentro do contexto de desenvolvimento de software, o termo release é abordado de várias maneiras, porém, a que mais se aproxima ao sentido real é que release é uma distribuição, pública ou privada, de uma primeira ou uma nova versão atualizada de um determinado software [20], portanto, sua aplicação consiste em planejar e organizar as distribuições de um software.

### 2.5.6 Build

Durante a entrega do software, o processo de compilação, teste e empacotamento do sistema é chamado de build, também conhecido como o processo de construção do software, sua principal função consiste em gerar um ou mais artefatos, com versões específicas para o deploy em produção [2]. Para a prática da integração contínua se faz importante possuir um build automatizado,

para que esse tipo de processo não tenho um custo de trabalho e tempo da equipe elevado [7], para que se torne uma prática constante durante o desenvolvimento do SW, e consiga assim antecipar e até mesmo evitar incompatibilidades operacionais.

## 2.6 Integração Contínua

A integração contínua é uma das práticas presentes na metodologia ágil, com origem na Extreme Programming (XP), ela tem como objetivos, encontrar e investigar bugs mais rapidamente, melhorar a qualidade do software e reduzir o tempo que leva para validar e lançar novas atualizações de software [09]. A integração contínua é uma prática de desenvolvimento de software na qual os membros de um time integram seu trabalho frequentemente. Usualmente, cada um integra sua parte pelo menos uma vez por dia, fazendo assim com que se tenha várias integrações junto à linha principal por dia. Para ser aprovada a integração é rodado um build de forma automatizada, onde são feitos os testes, gerados os relatórios e após isso é disponibilizado a versão, com isso é verificado de forma rápida e antecipada se existem erros antes da a integração à linha principal ser realizada [7]. Duvall ilustrou a forma como a integração contínua é realizada na prática, através da figura 3 podemos ver que os commits dos vários desenvolvedores, são centralizados em um repositório central, onde o servidor de integração contínua faz a operação de pull(puxar) no mesmo, e roda os scripts nos códigos desenvolvidos, gerando assim relatórios de feedback para a equipe de desenvolvedores consultarem e tomar decisões a cerca do que está sendo desenvolvido.

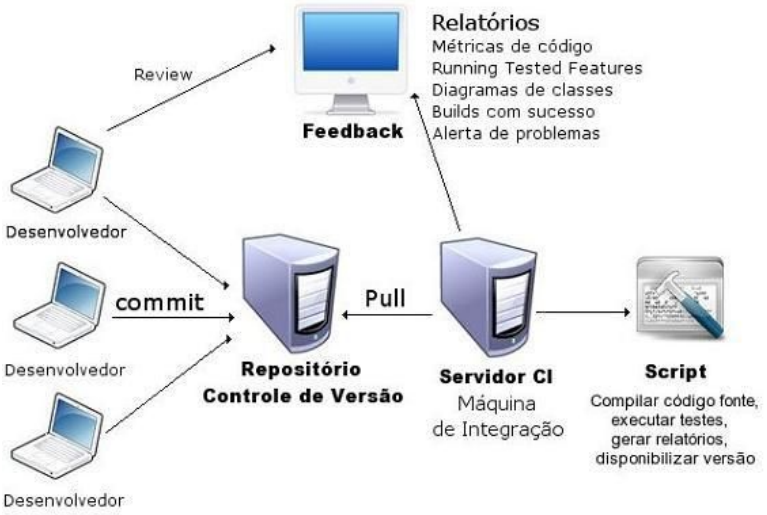



Figura 3: Processo de Integração Contínua [11]

Na IC, seu software passa por testes automatizados a cada mudança, assim o desenvolvedor sabe o momento em que ocorre um erro, podendo assim corrigi-lo imediatamente, a IC faz com que os times de desenvolvimento entreguem SW com maior velocidade, e com um menor número de erros, em relação as equipes que não a usam, isso faz com os custos e o tempo gasto no desenvolvimento seja reduzido [10] . Duvall listou 5 benefícios obtidos com a IC [11]:

- Redução dos riscos;
- Redução da execução manual de processos repetitivos;
- Geração de software entregável a qualquer momento;
- Melhorar a visibilidade do projeto;
- Ampliar a confiança da equipe no produto sendo desenvolvido;

Como pode ser visto IC se tornou uma prática importante para se desenvolver um software com mais rapidez, sem afetar a qualidade, pois, mesmo com o trabalho sendo realizado em equipe ocorre a redução de erros, pois como a integração é feita de forma automatizada, a detecção de erros também é, diminuindo assim o tempo para tal erro ser corrigido. Isso faz com que a IC se torne essencial para a implantação de DevOps [10].

## 2.7 Entrega Contínua

A entrega contínua é uma prática que surgiu visando diminuir tempo, custo e os riscos na entrega de software, ela nasceu como uma das prioridades do manifesto ágil [4]. Nela os times de ES devem produzir em ciclos de pequena duração, para que os SW possam ser lançados a qualquer momento e de forma confiável, visando assim testar e lançar o software aos usuários com maior velocidade e com maior frequência. EC é uma prática que garante a entrega de SW da equipe de desenvolvedores para o ambiente de produção em um processo confiável, previsível, visível e o mais automatizado possível, com riscos quantificáveis e bem entendidos [10].

Na EC a equipe de desenvolvimento foca em pequenas partes do projeto para a produção, ao invés de planejar fazer o release de grandes partes, isso faz com que sejam reduzidos os riscos de acontecer erros durante a implantação, além do mais na EC os testes manuais são trocados por testes automatizados, reduzindo assim o tempo e o custo de trabalho da equipe, por este motivo a IC é de grande importância para se conseguir implementar a EC [10].

Para que as atividades sejam transformadas em uma etapa normal e contìnua é fundamental que todos os membros da equipe trabalhem em conjunto, isso inclui desde desenvolvedores até times de implantação e operação [2]. Para se aplicar a EC os processos devem ser auxiliados por ferramentas e scripts de automatização que garantam que o software estará pronto para ser implantado em produção a qualquer momento [10].

## 2.8 Devops

Desde o surgimento do Manifesto Ágil [4], os problemas causados pela falta de trabalho colaborativo entre o time de desenvolvimento e o time de operações de uma organização ficaram cada vez mais presentes [18], muitos departamentos de TI possuem uma divisão clara de responsabilidades entre o time de desenvolvimento e o time de operações. No pensamento tradicional, o time de desenvolvimento é responsável por criar novos produtos e aplicações, adicionar funcionalidades ou corrigir bugs, já o time de operações tem a responsabilidade de cuidar desses produtos e aplicações em produção. O time de desenvolvimento é incentivado a introduzir mudanças, enquanto o time de operações preza pela estabilidade [2], essa distinção de responsabilidades gera vários conflitos entre os dois times, sendo assim, um dos principais problemas para o desenvolvimento ágil.

Essa ausência de colaboração entre os times de desenvolvimento e o de operações, afeta tanto a velocidade do lançamento de novas funcionalidades para seus usuários, quanto de manter estável seus sistemas operando com alta disponibilidade e serviços de qualidade[4] causando impacto diretamente nos negócios da organização, com mercados que ficam cada vez mais competitivos, em que os softwares devem se tornar cada vez mais rápidos e eficientes, tendo que entregar funcionalidades e realizar mudanças com uma frequência cada vez maior, além de ter que conseguir se conectar com uma imensa diversidade de dispositivos e sistemas operacionais, se ter um processo de qualidade durante todo o ciclo de vida do SW se tornou cada vez mais importante.

Em meados de 2008, durante o evento Agile, o tema metodologia ágil para a administração de infraestrutura começou a ser discutido, possuindo por base o desenvolvimento ágil de SW, a partir dessa discussão Patrick Debois criou um

encontro chamado *DevOpsDay*, visando discutir um novo tipo de cultura a ser inserida nas organizações de TI.

Assim surgiu o movimento DevOps, com um nome bastante intuitivo, realizou a junção do termo “Dev” de Development(desenvolvimento), com o termo “Ops” de Operations(Operações), foi criado para acabar com a idéia de que os times devem trabalhar em silos separados, e assim alinhar o time de desenvolvimento e o time de operações, consistindo em um conjunto de práticas destinadas a reduzir o tempo entre cometer uma mudança para um sistema e a mudança sendo colocada em produção normal, garantindo simultaneamente alta qualidade [5] e que visa melhorar a comunicação entre desenvolvedores e o time de operações para resolver problemas críticos, como o medo da mudança e as implantações de risco [6].

### 2.8.1 Pipeline Devops

São várias as práticas adotadas para se ter DevOps nas organizações, diversas variáveis são levadas em consideração para se decidir qual metodologia irá ser utilizada, porém, algumas práticas são bastante comuns nas organizações de TI que adotam Devops, e a pipeline de entrega é uma delas, por ser de suma importância para se ter uma melhor comunicação entre os times de desenvolvimento e de operações, e assim poder garantir uma entrega contínua, que por sua vez é essencial para a prática do Devops. A pipeline de qualquer organização pode ser personalizada da forma como necessitar, porém Humble e Farley ilustraram uma pipeline típica de Devops para se ter a garantia de entrega contínua.


Figura 2: Pipeline de entrega DevOps [11]

## 2.9 Considerações Finais

Neste capítulo foi apresentada a fundamentação teórica sobre o assunto, onde foram listadas as definições sobre os assuntos principais, DevOps e branching, além das outras técnicas usadas atualmente nas organizações de TI, e como eles estão inseridos nas atividades do desenvolvimento ágil de software. Essa fundamentação nos possibilitou a análise mais aprofundada da técnica de branching, e como a mesma impacta na cultura DevOps, sendo necessário o estudo da melhor forma de se realizar tal prática.

# 3. Metodologia

Foi realizada uma pesquisa onde através dos resultados obtidos, foram extraídos conhecimentos relacionados às práticas ocorrentes na indústria de software, este levantamento nos mostrou o comportamento e as práticas utilizadas para o uso de branching em organizações que utilizam DevOps e quais os anti-padrões existentes na literatura.

A pesquisa foi realizada através de um questionário online, a fim de explorar os dados dos entrevistados para serem analisados e transformados em conhecimento. As respostas do questionário foram alinhadas ao referencial teórico obtido com a análise do estado da arte realizada em relação ao uso de branching juntamente com seus anti-padrões e a cultura DevOps, e com isso conseguir levantar respostas para as hipóteses de pesquisa. Por fim vale ressaltar que o tema investigado é recente, portanto as questões foram elaboradas de forma que, as respostas fossem extraídas e interpretadas para serem comparadas com a literatura atual acerca do tema.

## 3.1 Questionário

Para fazer o levantamento das práticas existentes na indústria foi elaborado um questionário online, em português, onde foi divulgado em meios como, email, mensagem pessoal e pessoalmente, isso se deve ao fato de ser um pesquisa direcionada ao grupo de indivíduos que possuam cargos de gerência em empresas de software ou até mesmo desenvolvedores líderes de suas equipes. O questionário ocorreu durante o mês de outubro e novembro de 2017. A pesquisa foi desenvolvida com 23 perguntas, contendo questões abertas e subjetivas, sendo divididas em 4 sessões:

1. **Perguntas Pessoais:** Foi explorado o perfil do indivíduo que estaria dando aquelas resposta, a fim de extrair seu cargo e o tempo de experiência do mesmo.

2. **Relacionadas a organização:** Devido ao fato de DevOps ser definido como uma cultura organizacional, foi criada esta sessão com o intuito de extrair informações da organização como um todo onde as práticas aconteciam, nela foram extraídas informações como o tipo de organização, seu tamanho, sua maturidade e suas práticas relacionadas a desenvolvimento de software.
3. **Relacionadas ao time de desenvolvimento:** Para embasar ainda mais as informações da sessão anterior, esta seção foi criada para extrair informações mais específicas do time de desenvolvimento da organização.
4. **Gerenciamento de Configuração:** A maior seção da pesquisa, ela abrange um maior número de áreas por isso seu maior número de questões, sendo criada para extrair informações das práticas diárias da equipe de desenvolvimento, além de identificar os anti-padrões existentes relacionados a branching e as práticas de integração contínua, atividade essencial para o DevOps.

Vale ressaltar que os anti padrões listados na pesquisa e no trabalho foram extraídos do estudo '*Assessing the Value of Branches with What-if Analysis'* [27] onde foi realizada uma pesquisa para verificar se os anti padrões relacionados a branching ocorriam na Microsoft e qual o seu real impacto. Além disso foram extraídas do questionário aplicado neste trabalho, foram elas:

1. Aproximadamente, quantas horas por mês gasta em operações de ramificação como criar ramos e integrar mudanças de outros ramos?

2. Quanto tempo uma integração leva em média, incluindo o tempo para verificar se as mudanças foram integradas corretamente?

3. Como você valida a correção de uma integração?

4. Com base na sua experiência quantos erros são causados por integrações incorretas e em quais áreas esses erros ocorrem?

5. Com base na sua experiência, quantas vezes você encontrou os seguintes padrões anti-padrões?

## 3.2 Etapas de Pesquisa

Para alcançar os objetivos pretendidos a pesquisa foi aplicada a partir de algumas etapas, foram elas:

**Levantamento do Referencial teórico**: Nesta etapa foi realizada a busca dos principais trabalhos que abordassem os dois principais temas da pesquisa, Branching e DevOps, onde foi compreendido seus principais conceitos tanto teóricos como práticos. Nela foi possível identificar a real necessidade de uma pesquisa exploratória que abordassem os dois temas em conjunto.

**Extração dos resultados:** Nesta etapa as respostas obtidas no formulário online foram listadas, com isso facilitou a extração de informações através da mesma. Após a extração das informações, houve a comparação entre os trabalhos sobre branching em um contexto geral de desenvolvimento de software encontrados na literatura e as informações que foram identificadas como praticantes da cultura DevOps através das suas respostas.

**Análise dos Resultados:** Nesta etapa, os resultados alcançados foram descritos, mostrando o conhecimento adquirido através da pesquisa, para que assim fosse disponibilizado para trabalhos futuros.

## 3.3. Considerações Finais

Neste capítulo foram apresentados a metodologia para o desenvolvimento e a aplicação da pesquisa, através de formulário online. Detalhando cata etapa realizada, incluindo como as questões foram desenvolvidas, o que cada seção buscou investigar, e como foi realizada a análise e a extração de conhecimento através das respostas obtidas e a comparação com a literatura. No próximo capítulo será detalhada as hipóteses de pesquisa, juntamente com a análise crítica dos resultados obtidos através do formulário.

# 4. Análise dos Resultados

O presente capítulo identifica as hipóteses levantadas com relação às práticas relacionadas a cultura DevOps nas organizações e qual o tipo de impacto que essas práticas têm na técnica de branching em que os times de desenvolvimento dessas organizações utilizam.

## 4.1 Hipóteses de Pesquisa

Para a aplicação do formulário online foram definidas algumas hipóteses de pesquisa a serem respondidas [26]. Essas hipóteses foram definidas a partir dos cenários encontrados para o uso da técnica de branching e da presença da cultura DevOps nas organizações através das práticas de integração contínua, essencial para o DevOps.

- **HP01:** Times com poucos integrantes e que usam metodologia ágil não usam branching durante o desenvolvimento.
- **HP02:** As organizações que possuem a cultura DevOps são as que possuem menos idade.
- **HP03:** A maior incidência de anti padrões de branching ocorrem em projetos que possuem várias branchs no seu desenvolvimento.
- **HP04:** Quando muito tempo é utilizado para merge e para as operações relacionadas a branching a incidência de anti-padrões é maior.
- **HP05:**A prática de integração contínua extingue a possibilidade de anti-padrões de branching.

- **HP06:** Os líderes de times de desenvolvimento consideram grande o impacto que a técnica de branching tem na produtividade do time.

Para melhor observar como foi averiguada cada hipótese foi realizado um mapa da distribuição delas por seção do questionário on-line. A Figura 2 mostra as hipóteses de pesquisa mapeadas de acordo com cada subdivisão do questionário.

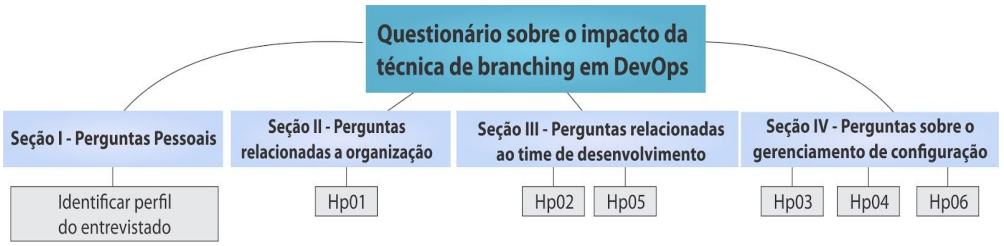



Figura 04: mapa de distribuição das hipóteses de pesquisa

## 4.2 Observação e Discussão dos Resultados

Neste capítulo, serão relatados os resultados obtidos com as perguntas do formulário on-line e realizada a análise com relação ao que foi concluído dos dados coletados. Nele será explanado o conhecimento adquirido com a pesquisa.

### 4.2.1 Perfil dos profissionais

Assim como descrito anteriormente, a pesquisa foi direcionada para profissionais da área de desenvolvimento de software, que possuem algum cargo de chefia ou liderança dentro do seu time, independente da função, com isso cargos que não tivessem esse perfil foram excluídos da parte de extração através das respostas. Desse modo a pesquisa foi divulgada diretamente para esses indivíduos através de emails e mensagens pessoais. Além disso foi verificado o tempo de experiência dos entrevistados, que resultou em uma média de 9,6 anos de experiência.

A Tabela 1 nos mostra os cargos dos entrevistados através do formulário online:

| Função | Quantidade | Tempo de experiência profissional (média) |
|---|---|---|
| Gerente | 9 | 11 anos |
| Líder técnico | 10 | 9 anos |
| Desenvolvedor | 4 | 4 anos |
| Analista/Engenheiro de Software | 9 | 8 anos |

Tabela 1: Cargos dos profissionais

Também foi observado em um primeiro momento algumas características das organizações em que os entrevistados atuam, isso se deve ao fato de DevOps ser uma cultura relativamente nova, por isso a necessidade de verificar em que contexto estão essas organizações que aplicam este tipo de cultura.

Dentre as características levantadas temos o porte em relação ao número de funcionários, divididos em quatro opções, Micro (até 9 funcionários), Pequeno porte (de 10 a 49 funcionários), Médio porte (de 50 a 99 funcionários) e Grande porte (Acima de 99 funcionários). O Gráfico 1 nos mostra a divisão dos tipos de organizações em que os entrevistados atuam, destacando uma aparente divisão.


Gráfico 1: Porte das organizações em que os entrevistados atuam

### 4.2.2 Uso de Branching

Foi identificado durante o planejamento da pesquisa que seria uma necessidade investigar qual o perfil dos times que utilizam branchs no seu desenvolvimento, para que assim pudesse ser identificado onde esta técnica estaria mais presente nas organizações que utilizam DevOps.

Como planejado as primeiras perguntas serviram para responder a primeira e a segunda hipóteses da pesquisa, a primeira afirmava que times com poucos integrantes e que utilizam metodologia ágil usam branching durante o desenvolvimento e a segunda que as organizações que possuem a cultura DevOps são as que possuem menos idade.

O questionário dividiu os times em 4 categorias, menos de 5 integrantes, entre 6 e 15 integrantes, entre 16 e 30 integrantes e acima de 30 integrantes sendo constatado a predominância por times entre 6 e 15 integrantes, como mostra o Gráfico 2.


Gráfico 2: Tamanho dos times de desenvolvimento

Para identificar o uso de metodologias ágeis nas organizações dos entrevistados foi realizada uma pergunta verificando sobre esse uso, onde 90,6% dos entrevistados confirmou o uso de metodologias ágeis no desenvolvimento de software, como mostra o Gráfico 3.

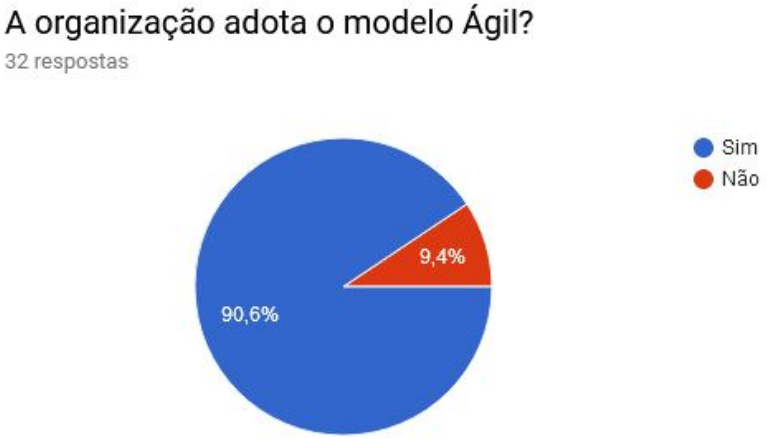



Gráfico 3: Times que adotam o modelo ágil.

Analisando conjuntamente as respostas dessas duas perguntas conseguimos responder a HP01. As respostas nos mostraram que 77,7% dos times com poucos integrantes (abaixo de 5 integrantes) utilizam metodologia ágil e 85,7% destes times utilizam a técnica de branching no desenvolvimento de software, confrontando a afirmação de Dantas (2006) em que o uso de branching aumentou significadamente em equipes com muitos integrantes, isso nos mostra que o aumento dessa técnica aumentou também em times pequenos mostrando que branching está inserido na indústria como um todo, independente do tamanho da equipe.

O Gráfico 4 nos mostra a presença de branching em times pequenos que utilizam métodos ágeis:

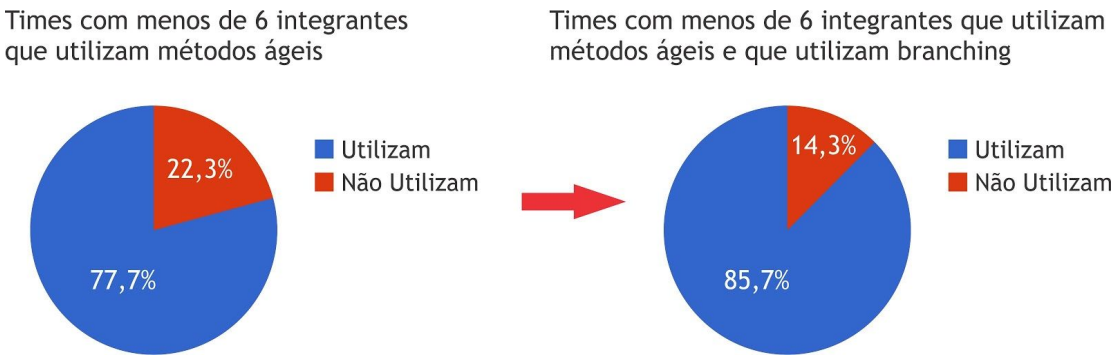



Gráfico 4: Uso de branching em times pequenos que utilizam métodos ágeis

### 4.2.3 Organizações que utilizam DevOps

Ainda nesta sessão foi verificado se organizações com pouca idade possuem a cultura DevOps, essa necessidade partiu da informação obtida de Rebellabs, em que o grande aumento do uso da cultura DevOps se dá por meio de empresas com pouco tempo de existência e que estariam mais abertas às mudanças propostas pelo movimento DevOps. Essa informação foi extraída do questionário online através das práticas de integração contínua, que segundo Humble e Farley, são práticas essenciais para a presença da cultura DevOps. Desse modo foi verificado que das 14 organizações que possuem servidor e práticas de integração contínua 12 são organizações com 10 ou mais anos de existência, isso nos mostra que práticas relacionadas a DevOps apesar de serem relativamente recentes estão inseridas desde as organizações mais novas, quanto as mais antigas, conforme apresentado no gráfico 5.


Gráfico 5: Organização com 10 ou mais anos e que possui IC

### 4.2.4 Anti-padrões no desenvolvimento através de branching

Segundo Bird e Zimmermman, projetos que possuem um número elevado de branchs com seus desenvolvedores, acabam por obter um maior número de anti-padrões [28] durante o seu uso, e isso foi o que procurou ser investigado e identificado pela HP03. Com isso fica ainda mais clara a necessidade de investigar se existe um número sugerido para evitar a incidência de anti-padrões de branching.

Através do questionário ficou evidente que projetos que possuem por natureza um número elevado de branchs (mais que 5 branchs) possuem uma grande possibilidade de ocorrerem anti-padrões durante o desenvolvimento, na pesquisa foi levantado que 90% dos projetos que possuem muitas branchs ocorrem os anti-padrões definidos por appleton et al. Porém ao analisarmos os projetos que possuem um número baixo de branchs (5 ou menos branchs), conseguimos verificar que o número de incidência dos anti-padrões é menor porém continuam ocorrendo. O gráfico 06 nos apresenta a quantidade de anti-padrões de branching em projetos tanto com mais de 5 branchs, quanto com menos de 5 branchs.

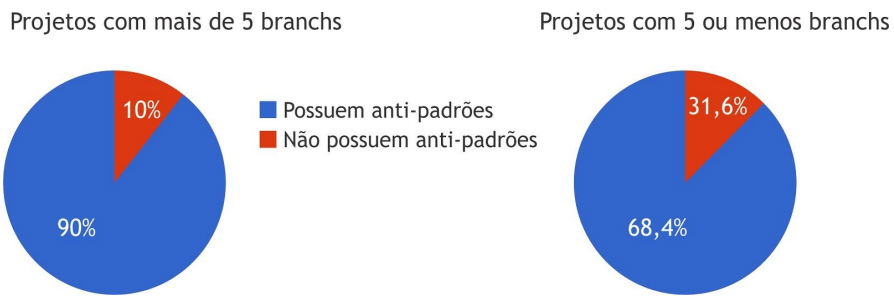



Gráfico 06: Incidência de anti-padrões de branching

O tempo é outra dimensão que pode nos mostrar quando um projeto poderá ter os anti-padrões de branching [27]. Através do questionário foi possível visualizar a quantidade de tempo gasto com operações de branching e com o merge durante o desenvolvimento de cada organização e compará-los com as respostas a respeito dos anti-padrões [28] encontrados.

Através das respostas podemos perceber que em 66,6% dos casos em que é gasto muito tempo (acima de 6 minutos) com a operação de merge, acontece do time de desenvolvimento gastar muito tempo mensal com as operações de branching (acima de 3 horas por mês), além disto podemos observar deste conjunto que em 100% dos casos ocorrem anti-padrões de branching durante o desenvolvimento afirmando o que Herskisov disse em sua pesquisa, vale salientar que o anti-padrão mais encontrado com 66,6% é o Merge Paranóia, onde a operação de merge é evitado a todo o custo, geralmente devido ao medo das conseqüências, por se ter gastado muito tempo para fazer o merge, se tornando uma tarefa que vai trazer complicações para o time de desenvolvimento [28].

Vale salientar que todos os projetos que gastam muito tempo com a operação de merge e com operações de branching não possuem ações de integração contínua, prática definida como essencial para a cultura DevOps [10]. Através da pesquisa pudemos verificar a diferença no tempo gasto com merge e com operações de branching em projetos que utilizam ou não a integração contínua. O Gráfico 07 nos mostra a média de tempo gasto, ficando evidente a diferença do tempo gasto com e sem integração contínua.

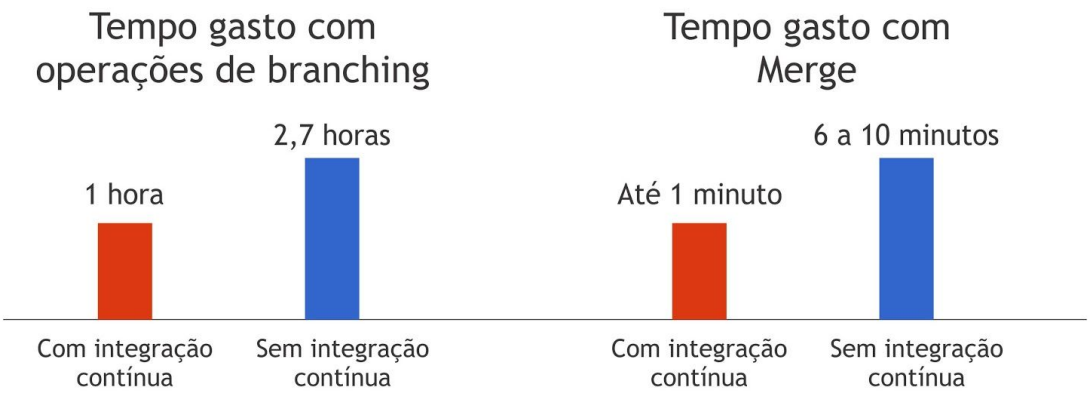



Gráfico 7: Diferença de tempo gasto com e sem integração contínua

### 4.2.5 DevOps e os anti-padrões de branching

A técnica de branching vem sendo cada vez mais usada em times de desenvolvimento, por trazer a possibilidade de vários desenvolvedores poderem trabalhar em paralelo sem afetar a linha principal do código, e assim aumentar a velocidade de desenvolvimento, reduzindo o risco de comprometer a operação do SW [15], porém, seu sucesso é questionado por alguns autores quando falamos de IC, que por sinal é um dos pilares para se introduzir a cultura do *DevOps* em qualquer organização. Humble e Farley afirmaram que usar *branching* é o contrário a IC, pelo fato de que a integração entre os *branchs* individuais não poderia ser assegurado ao ser utilizado este tipo de técnica, pois as mudanças realizadas individualmente não estariam disponíveis para os outros desenvolvedores, gerando assim o “inferno de fusão”, nome dado aos erros percebidos na hora da integração dos códigos. Porém, Schneid definiu que se for reduzido o tempo em que os *branchs* são integrados a linha principal, há a possibilidade de ser feito a IC, pois com *branchs* de curta duração, com prazo máximo de alguns dias, seria diminuído o tamanho do “inferno de fusão”, além do mais uma estratégia de branching mal

utilizada pode acarretar na grande incidência dos anti-padrões definidos por Appleton et. al. Porém, uma estratégia correta de branching traz ganhos significativos na produção do time, isso fica ainda mais evidente na pesquisa realizada por Bird e Zimmermann quando mudaram a estratégia de Branching na microsoft e tiveram ganhos significativos na produção [27].

Para verificar se práticas DevOps, como a integração contínua [10], causariam impacto positivo sobre branching foi verificado no questionário qual a porcentagem de incidência dos anti-padrões em projetos que utilizam essas práticas. Na pesquisa foram identificados 14 times que possuem tanto um servidor de integração contínua, como as atividades necessárias para o seu ciclo, como build, execução automática de testes unitários e ferramenta de análise de código [7], desses 5 responderam não ter ocorrência de anti-padrões durante o desenvolvimento, porém, dos outros 9 que afirmaram ter anti-padrões, 5 deles indicaram a incidência de um anti-padrão em específico, que é o Never-Ending Merge, definido por Appleton et. al. como uma atividade de merge que nunca termina, pois sempre há o que ser integrado. Isso destaca a necessidade de um estudo mais direcionado e aprofundado sobre a incidência deste tipo de anti-padrão em times que utilizam a técnica de branching em conjunto com a integração contínua.

### 4.2.6 Impacto de Branching no desenvolvimento de software

Assim como na pesquisa realizada por Bird e Zimmermann, verificamos a necessidade de averiguar a opinião pessoal dos entrevistados, pois além de verificar o referencial teórico a respeito do assunto, a parte prática na indústria é de suma importância na afirmação ou não das informações encontradas na literatura [29]. Com isso em uma das questões realizadas no formulário online perguntamos aos líderes entrevistados, como eles avaliam o grau de impacto dos anti-padrões

desta técnica na produção do time, foi verificado que a grande maioria dos entrevistados consideram um impacto de proporções grande ou moderada neste tipo de técnica, isso fica ilustrado no Gráfico 8 abaixo.

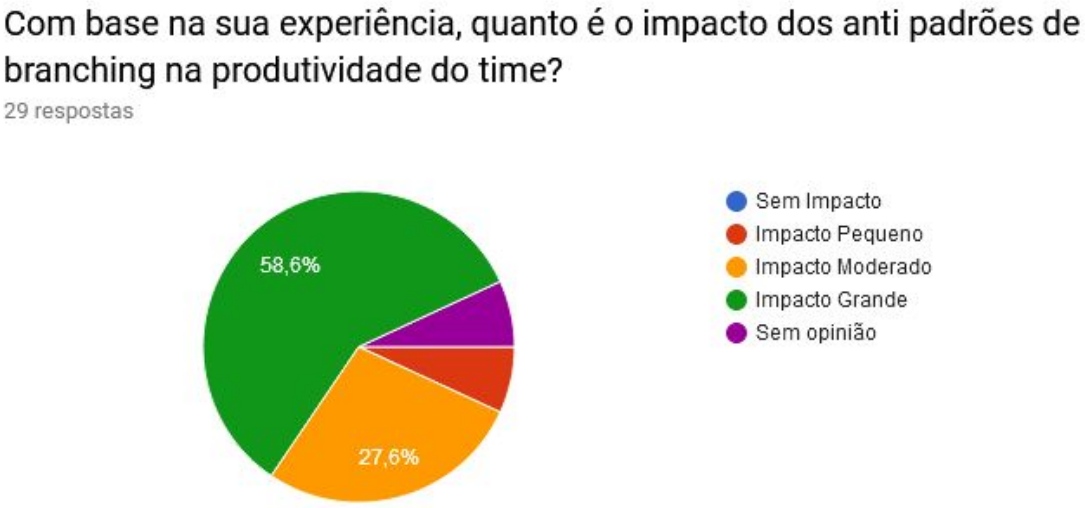



Gráfico 8: Impacto dos anti-padrões de branching

Isso torna ainda mais evidente a relevância da pesquisa realizada, mesmo que de forma mais geral, mostrando a necessidade de maiores pesquisas a respeito do assunto.

## 4.3 Discussão

Ao realizarmos uma análise geral do questionário online aplicado, pudemos observar vários indícios ligados a branching presentes na indústria de TI atualmente. Entre eles, podemos destacar o grande número de organizações que possuem a técnica de branching no seu desenvolvimento, destacando que, o uso desta técnica não está diretamente ligada ao tamanho da empresa, nem ao tempo da mesma no mercado, como afirmado por alguns autores. Desse modo, ficou mais fácil identificar alguns fatores que contribuem para o sucesso ou não do uso deste tipo de técnica, alinhada as práticas presentes na cultura DevOps, essa facilidade ocorreu por conta da identificação dos anti-padrões de branching definidos por

Appleton et al. [28], em que a incidência dos mesmos varia de acordo com as práticas do time de desenvolvimento.

Dentre as principais práticas utilizadas que foram identificadas na pesquisa e que contribuem para o sucesso do uso da técnica de branching alinhadas a práticas da cultura DevOps podemos citar:

1. **Uso de uma estratégia formal de Branching**

   O uso de uma estratégia formal de branching influencia muito no sucesso do uso desta técnica, através do questionário foi possível identificar que nos casos em que a incidência dos anti-padrões é menor, o time de desenvolvimento estava usando uma estratégia formal de branching, e não utilizado a mesma de qualquer modo, sem políticas e regras do uso das branchs dentro do projeto. Destacando-se o grande uso da estratégia de *Gitflow* por parte das equipes que usam uma estratégia formal de branching, e do sucesso atrelado ao uso desta estratégia.

2. **Utilização de Metodologia Ágil**

   O aumento na utilização da técnica de branching também está atrelada ao crescimento do uso de metodologias ágeis no desenvolvimento de software, isso ficou ainda mais evidente com a análise das respostas do questionário, onde todos os times identificados com um baixo número de anti-padrões encontrados utilizavam uma metodologia ágil no seu desenvolvimento. Isso acontece por conta do mundo do desenvolvimento de software ser repleto de mudanças constantes, como mudanças no código sendo solicitadas em espaços curtos de tempo, e essas mudanças tendo que ser gerenciadas em um desenvolvimento distribuído.

3. **Utilização de Sistemas de Controle de Versão**

   Uma prática essencial para o uso de branching é a adoção de um sistema de controle de versão, e atividades bem definidas para o uso desse sistema é de suma importância para o sucesso do uso de branching, pois é através desse sistema que será feito todo o gerenciamento das branchs. Na

pesquisa foi identificado que todos os projetos com baixo número de anti-padrões utilizam o Git como seu sistema de controle de versão, isso se dá pelo fato da facilidade do Git de trabalhar com branching, pois o mesmo além de possuir todo o histórico, possui várias operações definidas para serem feitas as ações nas branchs,

4. **Utilizar 5 ou menos Branchs**

Através do questionário foi possível identificar que o número de branchs causa impacto na incidência de anti-padrões, apesar de não ser um fator crítico do sucesso da estratégia ela possui influência na sua aplicação. Foi constatado que um número elevado de branchs aumenta a ocorrência de anti-padrões, isso se dá pelo fato do número elevado de branchs impactar diretamente na operação de merge, sendo uma porta de entrada fácil para a ocorrência dos anti-padrões. Desse modo, a fim de evitar problemas, utilizar um número baixo de branchs acaba sendo uma boa prática.

5. **Integração Contínua**

Apesar da definição de Humble e Farley de que a técnica de branching é contrária a integração contínua, a pesquisa nos mostrou respostas opostas a isso, através do questionário pudemos perceber que os times que não possuem ou têm um baixo número de anti-padrões existentes possuem práticas de integração contínua, mais precisamente todas as atividades e ferramentas necessárias para essas práticas como um servidor de integração contínua além de ferramentas de build (Ex: Maven, ant, gradle, make), de execução automática de testes unitários (Ex: jUnit, utPLSQL, Cunit), e ferramentas de análise de código (Ex: SonarQube).

Desse modo fica claro o impacto que branching e seus anti-padrões causam no desenvolvimento de software, inclusive nas práticas DevOps, por ser uma cultura que preza pelo ambiente de trabalho colaborativo, e que da foco a velocidade e a automação, acaba tendo muitos ganhos com o uso da técnica de branching. Dessa

forma é imprescindível possuir boas práticas relacionadas a esta técnica para que os objetivos esperados sejam alcançados.

## 4.4 Considerações Finais

Neste capítulo foi apresentado e descrito os resultados encontrados a partir do questionário online realizado, com isso tornou claro o impacto que a técnica de branching e seus anti-padrões causam em práticas de desenvolvimento de software, entre elas o DevOps, podemos observar que seu uso deve ser bastante planejado ne alinhado com práticas de integração contínua, pois ela causa um forte efeito no desenvolvimento, sendo percebido a realização de trabalhos futuros mais aprofundados, com a observação real dos temas estudados na indústria de TI.

# 5. Conclusões

De acordo com o trabalho, pudemos observar que a Cultura DevOps está cada vez mais presente na indústria de TI, tanto nas empresas mais novas, como nas mais antigas, além disso foi constatado o alto número de organizações que utilizam metodologias ágeis para o seu desenvolvimento, destacando também o número elevado de organizações que utilizam a técnica de branching nas suas atividades. Porém, foi constatado que o relacionamento entre Branching e DevOps é algo bastante novo, a mesma nos mostrou que pelo motivo desta cultura ser relativamente recente, ainda não foram feitos estudos à cerca da junção desta técnica de desenvolvimento neste tipo de cultura organizacional, não sendo uma exclusividade da técnica de Branching.

Também foi possível verificar que a incidência de anti-padrões da técnica de branching é algo bastante corriqueiro na indústria de desenvolvimento de software e que essa presença pode ser atrelada a uma série de indícios presentes durante o dia a dia do time de desenvolvimento, destacando-se os números, onde projetos que possuem um número alto de branchs (acima de 5) acabam gastando muito tempo com a operação de merge (acima de 6 horas) e com as operações de branching (2,7 horas por mês) levando a incidência de anti-padrões. Além desses indícios foi possível compreender que a ausência de práticas e ferramentas da cultura DevOps utilizadas pelo time também influenciam na incidência de anti-padrões.

Além disso, foi verificado que os times que possuem menos anti-padrões são os times de desenvolvimento que utilizam atualmente integração contínua, prática essencial a cultura DevOps, sendo verificado que as práticas e ferramentas relacionadas a esta cultura possuem influência positiva na diminuição do surgimento dos anti-padrões durante o desenvolvimento, além do ganho de tempo para o time,

onde times que possuem a cultura DevOps gastam menos tempo com operações de merge e operações relacionadas a branching, resultando em aumento de produção. Sendo constatado também que este fato já recebe bastante atenção dos líderes dos times de desenvolvimento, que consideraram na pesquisa como grande, o impacto causado por branching no desenvolvimento de software.

Através deste trabalho foram verificadas sete hipóteses de pesquisa desenvolvidas e verificadas através do formulário on-line. Essas hipóteses deveriam ser confirmadas ou refutadas [29]:

- **HP01:** Times com poucos integrantes e que usam metodologia ágil não usam branching durante o desenvolvimento.

  Essa hipótese foi refutada, através do questionário foi visto que 77,7% dos times com pouco integrantes utilizam a metodologia ágil e 85,7% destes times utilizam a técnica de branching.

- **HP02:** As organizações que possuem a cultura DevOps são as que possuem menos idade.

  Outra hipótese refutada, pois foi verificado nas respostas de organizações que adotam a cultura DevOps, possuem as mais variadas idades, mostrando que apesar de ser uma cultura nova, está cada vez mais presente nas organizações, independente da idade..

- **HP03:** A maior incidência de anti padrões de branching ocorrem em projetos que possuem várias branchs no seu desenvolvimento.

  Através da pesquisa essa hipótese pode ser confirmada, pois 90% dos projetos que possuem mais de 5 branchs tem uma grnade incidência de anti-padrões no seu desenvolvimento, enquanto a incidência em projetos com menos de 5 branchs é de 68,4%.

- **HP04:** Quando muito tempo é utilizado para merge e para as operações relacionadas a branching a incidência de anti-padrões é maior.

  Esta hipótese foi confirmada através da pesquisa, com um valor de bastante destaque, foi verificado que em 100% dos projetos que gastam muito tempo para realizar o merge e para realizar as operações de branching são encontrados anti-padrões durante o desenvolvimento..

- **HP05:**A prática de integração contínua extingue a possibilidade de anti-padrões de branching.

  A hipótese foi refutada ao ser verificado que projetos com integração contínua também possuem anti-padrões de branching, porém, foi verificado que quase a metade dos projetos não possuem anti-padrões, sendo já um número bastante expressivo, mas que não extingue a possibilidade de anti-padrões.

- **HP06:** Os líderes de times de desenvolvimento consideram grande o impacto que os anti-padrões da técnica de branching tem na produtividade do time.

  Através da pesquisa pode-se perceber que os líderes dos times de desenvolvimento já estão atentos ao impacto que está técnica pode causar no desenvolvimento da equipe, na pesquisa foi levantado que 58,6% dos entrevistados consideram grande o impacto dos anti-padrões de branching no desenvolvimento do time.

## 5.1 Limitações do Trabalho

As limitações do trabalho incluem uma série de itens, como escassez de insumo, onde os que existem, o acesso não é possível, além de que é um tema novo e que não concentrou estudos voltados para esse tipo de análise e seria necessário uma pesquisa de campo mais aprofundada para levantar dados de uso deste tipo de técnica no mundo atual onde o DevOps está cada vez mais presente.

## 5.2 Trabalhos Futuros

- Pesquisa de campo na Indústria de TI para levantamento de dados reais ligados ao tema
- Busca de casos de uso reais relacionados ao tema
- Comparação da técnica de branching em relação às outras técnicas
- Definição de modelo sugerido para a aplicação real na indústria de TI
- Averiguar a correlação entre as respostas encontradas na pesquisa

## Referências


[1] PRESSMAN, R. **Engenharia de Software: uma abordagem profissional.** 8ª Ed. Germany, 2016.

[2] SATO, D. **Devops na Prática: Entrega de software confiável e automatizada**. São Paulo, 2013. ISBN 9788566250404.

[3] SOMMERVILLE, I. **Engenharia de Software***.* 9 Ed. São Paulo, 2011.

[4] BECK, K. et al., **Manifesto for Agile Software Development***.* 2001*.* Disponível em <https://agilemanifesto.org/> Acessado em 09 abr 2017.

[5] BASS, L.; WEBER, I.; ZHU, L. **Devops: A software architect's perspective**. United States: Addison-Wesley Educational Publishers, 2015. ISBN 9780134049847.

[6] HüTTERMANN, M. **DevOps for developers**. New York: Springer-Verlag New York, 2012. ISBN 9781430245698

[7] FOWLER, M. **Continuos Integration***. 2006.* Disponível em <https://martinfowler.com/articles/continuousIntegration.html> Acessado em 06 abr 2017.

[8] REBELLABS. **IT Ops & DevOps Productivity Report 2013: Tools, Methodologies and People**. 2013.

[09] AMAZON WEB SERVICES. **O que significa integração contínua?.** Disponível em <https://aws.amazon.com/pt/devops/continuous-integration/> Acesso em 06 abr 2017

[10] HUMBLE, J.; FARLEY, D. **Continuous Delivery: Reliable software releases through build, test, and deployment automation**. United States, 2010. ISBN 9780321601919.

[11] DUVALL, Paul. **Continuous Integration: Improving Software Quality and Reducing Risks**. Addison-Wesley, 2007.

[12] TELES V. M. **Integração contínua**. Disponível em <http://www.desenvolvimentoagil.com.br/xp/praticas/integracao> Acesso em 19 jun 2017.

[13] GARCIA, Francisco **A. Integração Contínua: da teoria à prática – Parte 1**. Java Magazine, Rio de Janeiro, n. 117, 2013. Disponível em: <http://www.devmedia.com.br/integracao-continua-da-teoria-a-pratica/28284> . Acesso em 19 jun 2017.

[14] DANTAS, Cristine. **Gerência de configuração de Software**, Java Magazine, Rio de Janeiro, 2006. Disponível em: <http://www.devmedia.com.br/gerencia-de-configuracao-de-software/9145> Acesso em 19 jun 2017.

[15] I. HERSKOVTIS. **Source Control Branching for Success**. Disponível em <https://dzone.com/articles/source-control-branching-for-success>. Acesso em 19 jun 2017.

[16] FILHO A. B. Et al. **Gerência de Configuração: Definições Iniciais, Ferramentas e Processo .** 2010.

[17] RAMOS. E. S., FREITAS. R. C. **Análise Comparativa de Sistemas de Controle de Versões Baseados em Código Aberto .** 2013..

[18] SCHNEID, K. **Branching strategies for developing new features within the context of continuos delivery**. 2017.

[19] E. CHARLES**. O que significam e quais as melhores práticas para usá-los?.** Disponível em <https://pt.stackoverflow.com/questions/20989/o-que-branch-tag-e-trunk-realmente-significam>. Acesso em 27 jun 2017.

[20] WIKIPÉDIA. **Liberação de Software** Disponível em**:** https://pt.wikipedia.org/wiki/Libera%C3%A7%C3%A3o_de_software>. Acesso em 27 jun 2017.

[21] P. BOURQUE, R. FAIRLEY, eds., **Guia do Conjunto de Conhecimento de Engenharia de Software***,* IEEE Computer Society, 2014 .Disponível em: <https://www.computer.org/web/swebok/v3 > Acesso em 02 Jun 2017.

[22] MEIRA S. Et al. **Programming the Universe: The First Commandment of Software Engineering for all Varieties of Information Systems.** 2016.

[23] S. DENNING**. What Is Agile**?. Disponível em <https://www.forbes.com/sites/stevedenning/2016/08/13/what-is-agile/#2b582ca726e3> Acesso em 29 Jun 2017.

[24] REBELLABS. **IT Ops & DevOps Productivity Report 2016: Featuring data on IT performance, employee engagement, ROI and more** . 2016.

[25] BABICH, W. A., **Software Configuration Management,** Addison-Wesley, 1986.

[26] GIL, ANTÔNIO CARLOS. **Métodos e Técnicas de Pesquisa Social**. 6ª Ed. São Paulo: Atlas, 2008.

[27] BIRD C., ZIMMERMANN T. **Assessing the Value of Branches with What-if Analysis,** 2012.

[28] Appleton, B., Berczuk, S., Cabrera, R., and Orenstein, R. Streamed Lines: **Branching Patterns for Parallel Software Development.** In The Pattern Languages of Programs Conference ,1998.

[29] Wazlawick, R. S. **Metodologia de pesquisa para Ciência da Computação.** 2.ed.: Elsevier, 2014.

[30] **Noções Básicas de Git**. Disponível em <https://git-scm.com/book/pt-br/v1/Primeiros-passos-No%C3%A7%C3%B5es-B%C3%A1sicas-de-Git> Acesso em 29 Nov 2017.

[31] **Learning Branching With Bitbucket Cloud**. Disponível em <https://www.atlassian.com/git/tutorials> Acesso em 29 Nov 2017.

# Apêndice

## Seção 1

## Seção 2

### Perguntas relacionadas a organização

Descrição (opcional)

Qual o tamanho da organização em que você trabalha? *

- ○ Grande porte (Acima de 99 funcionários)
- ○ Médio Porte (50 a 99 funcionários)
- ○ Pequeno Porte (10 a 49 funcionários)
- ○ Micro (Até 09 funcionários)

A organização é uma startup?

- ○ Sim
- ○ Não

Qual o ano de criação da organização? *

A organização adota o modelo Ágil? *

- ○ Sim
- ○ Não

Se a resposta anterior foi sim, qual(is) o(s) método(s) utilizado(s)?

- ☐ XP
- ☐ Scrum
- ☐ Kanban
- ☐ Não sei
- ☐ Outros...

## Seção 3

### Perguntas relacionadas ao time de desenvolvimento

Descrição (opcional)

Qual o tamanho do time de desenvolvimento que você lidera ou faz parte atualmente? *

- ○ Menos de 5 integrantes
- ○ Entre 6 e 15 integrantes
- ○ Entre 16 e 30 integrantes
- ○ Acima de 30 integrantes

Qual é a arquitetura utilizada no desenvolvimento do projeto em que você está alocado atualmente? *

- ☐ Orientada a serviços (Microserviços, SOA)
- ☐ Monolítico

# Seção 4

## Perguntas sobre Gerenciamento de Configuração

Descrição (opcional)

A organização utiliza algum sistema de Controle de Versão? *

- Sim
- Não

Se a resposta anterior foi sim, qual o sistema de controle de versão é utilizado?

Texto de resposta curta

O desenvolvimento é realizado através de branch (ex. feature branch, dev branch, release branch, hotfix )? *

- Sim
- Não (usamos apenas o baseline ou master)

Você adota alguma estratégia formal de branching (ex. gitflow) (https://www.atlassian.com/git/tutorials/comparing-workflows)

- Sim
- Não
- Outros...

Se a resposta anterior foi sim, qual é a estratégia adotada?

Texto de resposta curta

Qual a quantidade média de branchs existentes por aplicação?

Texto de resposta curta

Aproximadamente, quantas horas por mês é gasto em operações de branching, como criar branchs e integrar as mudanças de outras branchs? (Ex: 1,5 horas)

Texto de resposta curta

Quanto tempo em média leva o processo de integração (merge)?

- ○ até 1 minuto
- ○ de 2 a 5 minutos
- ○ de 6 a 10 minutos
- ○ de 11 a 20 minutos
- ○ Acima de 20 minutos

Qual(ais) o (s) anti-padrões acontece(m) no seu processo de branching? (Fonte: http://www.bradapp.com/acme/branching/)

- ☐ Merge Paranoia - O merge é evitado a todo o custo, geralmente devido ao medo das conseqüências.
- ☐ Merge Mania - Muito tempo gasto no código do merge, ao invés de desenvolvê-lo.
- ☐ Big Bang Merge - O merge é adiado e depois é feito o merge de todas as branchs simultaneamente.
- ☐ Never-Ending Merge - Atividade contínua de merge, pois sempre há o que ser integrado.
- ☐ Wrong-Way Merge - O merge é feito na branch errada.
- ☐ Branch Mania - São criados muitas branchs.
- ☐ Cascading Branches - As branchs nunca são integradas de volta a linha principal do código.
- ☐ Mysterious Branches - Branchs são criadas sem motivo aparente.
- ☐ Runaway Branches - Branch com finalidade única evolui para ser uma branch com várias finalidades não relacionadas
- ☐ Volatile Branches - Branchs com arquivos instáveis são integrados a outros ramos.
- ☐ Development Freeze - As atividades de desenvolvimento são paradas para o branching e merging serem realizados.
- ☐ Integration Wall - Em vez das branchs servirem para dividir os trabalhos, acabam dividindo os membros das equipes de
- ☐ Spaghetti Branching - Se integra mudanças entre branchs não relacionadas.

Com base na sua experiência, quanto é o impacto dos anti padrões de branching na produtividade do time?

- ○ Sem Impacto
- ○ Impacto Pequeno
- ○ Impacto Moderado
- ○ Impacto Grande
- ○ Sem opinião

Como é feita a validação do código antes da integração? (Ex: Antes de integrar, usamos um analisador automático.)

- ○ Peer Review com outro desenvolvedor
- ○ Peer Review com o líder técnico
- ○ Automática a cada commit
- ○ Semi-automática (Com aprovação do líder técnico)

Você adota algum servidor de Integração Contínua (ex. Jenkins, Travis, circleci)? *

- ○ Sim
- ○ Não

Qual(is) atividade(s) compõe(m) seu ciclo de Integração contínua?

- ☐ Build (Ex: Maven, ant, gradle, make)
- ☐ Execução automática de testes unitários (Ex: jUnit, utPLSQL, Cunit)
- ☐ Análise de Código (Ex: SonarQube)

Qual a principal causa de erros nas interações do seu servidor de Integração Contínua?

Texto de resposta longa