UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

RENNAN FRANCISCO MESSIAS DE LIMA

IMPLEMENTAÇÃO DE PIPELINE DE ENGENHARIA DE INTEGRAÇÃO E ENTREGA CONTÍNUA NO IFAC

Recife

2022

RENNAN FRANCISCO MESSIAS DE LIMA

IMPLEMENTAÇÃO DE PIPELINE DE ENGENHARIA DE INTEGRAÇÃO E ENTREGA CONTÍNUA NO IFAC

Dissertação apresentada ao Programa de Pós-Graduação em Ciências da Computação do Centro de Informática da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Mestre em Ciências da Computação.

Área de Concentração: Sistemas de Informação

Orientador (a): Prof. Dr. Vinicius Cardoso Garcia

Recife

2022

Catalogação na fonte
Bibliotecária Nataly Soares Leite Moro, CRB4-1722

L732i Lima, Rennan Francisco Messias de
Implementação de pipeline de engenharia de integração e entrega contínua no IFAC / Rennan Francisco Messias de Lima. – 2022.
238 f.: il., fig., tab.

Orientador: Vinicius Cardoso Garcia.
Dissertação (Mestrado) – Universidade Federal de Pernambuco. CIn, Ciência da Computação, Recife, 2022.
Inclui referências e apêndices.

1. Sistemas de Informação. 2. Integração contínua. 3. Entrega contínua. 4. Time to market. 5. Órgão público. I. Garcia, Vinicius Cardoso (orientador). II. Título

681.3 CDD (23. ed.) UFPE - CCEN 2022 – 185

Rennan Francisco Messias de Lima

Implementação de Pipeline de Engenharia de Integração e Entrega Contínua no IFAC

Dissertação apresentada ao Programa de Pós-Graduação Profissional em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Mestre Profissional. Área de concentração: Sistemas de Informação.

Aprovado em: 29/07/2022.

BANCA EXAMINADORA

_____

Prof. Vinicius Cardoso Garcia
Centro de Informática / UFPE
(Orientador)

_____

Prof. Lincoln Souza Rocha
Universidade Federal do Ceará

_____

Prof. Leandro Marques do Nascimento
Universidade Federal Rural de Pernambuco

À minha esposa que esteve ao meu lado em todos os momentos dessa etapa e a minha mãe pelo apoio para realização de mais um sonho.

AGRADECIMENTOS

A conclusão deste trabalho não seria possível sem a participação de pessoas especiais que fazem parte da minha vida, que por meio de manifestações de apoio, carinho e compreensão, mantiveram-me motivado a concluir esta etapa importante em minha vida profissional. Assim, expresso minha gratidão:

Primeiramente, agradeço a Deus pelo dom da vida, pela saúde e por me fortalecer nos momentos mais difíceis dessa longa caminhada iluminando cada passo dado;

À minha esposa Helen pelo carinho, incentivo, ajuda e compreensão durante minhas ausências nos períodos de viagens e estudos;

À minha mãe Nelir pelo apoio e orações. Professora de profissão, tive em casa as primeiras orientações sobre a importância e disciplina nos estudos;

Aos meus pais Raimundo e Moizaniel pelo exemplo dado de sempre querer conquistar meus sonhos;

Aos professores do Centro de informática da Universidade Federal de Pernambuco pelos conhecimentos repassados no decorrer das disciplinas, aos demais servidores e em especial a Joelma que sempre nos atendia com muita atenção e um sorriso no rosto.

Aos colegas da turma de mestrado 2018.2 que tive a oportunidade de conhecer e em especial aos companheiros de idas e vindas a Recife, Iago e Woshington que sem os quais essa jornada seria bem mais difícil;

Ao IFAC e Secretaria de Educação Profissional e Tecnológica (SETEC) pelo apoio durante as semanas de aulas e custeio do programa de mestrado;

Aos colegas da DSGTI do IFAC, pelos conselhos, sugestões, apoio e compromisso assumido durante minha ausência. Em especial aos colegas Jonas, Jair e Ricardo por me auxiliarem de diversas formas nas etapas da pesquisa;

Ao Professor e Orientador Dr. Vinicius Cardoso Garcia, por acreditar na ideia, pela parceria e por estar sempre presente e disponível em todas as etapas desta pesquisa;

A todos que contribuíram, indiretamente, para a realização desta pesquisa.

Muito obrigado!

"Não é a força, mas a perseverança que realiza grandes coisas." (JOHNSON; MURPHY, 1825, p. 15).

RESUMO

Nas instituições públicas federais de educação, a cada ano a população estudantil cresce, juntamente com o número de funcionários, gerando a necessidade de novos recursos tecnológicos como, por exemplo, sistemas de informação mais eficientes e com maior qualidade. Devido as equipes de desenvolvimento serem pequenas, com pouca maturidade em processos, ocorrem atrasos para o atendimento às demandas, além do processo de *build* e entrega do produto de software ser custoso e não repetível, ocasionado pelo excesso de erros, demora para fazer a entrega e a implantação, acarretando em correções frequentes, fazendo com que a entrega demore horas ou dias sem garantia da qualidade do produto. Isso acontece por conta do alto número de erros que são reportados tardiamente, gerando atrasos e consequentemente a perda dos prazos. Este trabalho teve como objetivo realizar um estudo sobre a adoção da integração e entrega contínua quanto ao ciclo de entrega do desenvolvimento de software no Instituto Federal do Acre (IFAC), de forma a verificar o impacto dessa adoção no processo de entrega e qualidade do produto. Para a realização desse estudo, foi executada uma revisão sistemática da literatura (RSL) que analisou 54 estudos publicados entre 2016 e 2019, para coletar evidências sobre desafios enfrentados, melhores práticas e ferramentas utilizadas na adoção e uso de integração e entrega contínua. Durante a revisão sistemática, foram encontrados 32 desafios, 20 melhores práticas e 63 ferramentas. A partir das evidências coletadas, foi construída uma proposta de abordagem baseada em evidências para o processo de adoção e gestão de riscos de um pipeline de integração e entrega contínua, através da relação e combinação de desafios e soluções (melhores práticas e ferramentas). Como validação da abordagem, ela foi aplicada em um projeto piloto do IFAC, sendo avaliada quanto à viabilidade, através do GQM, após a adoção, utilizando o método de comparação do projeto irmão. As principais contribuições deste trabalho foram: (1) a criação de um corpo de conhecimento, a partir das evidências coletadas como resultado da revisão sistemática, que funciona como um *framework* para compor o pipeline concreto e efetivo a ser implementado; (2) um processo de adoção de pipeline; e (3) uma abordagem para o gerenciamento de risco utilizando o corpo de conhecimento. Por fim, o processo criado nesta pesquisa foi aplicado em um projeto piloto do IFAC e documentado através de um relato de experiência apresentando estratégias mínimas para adotar o pipeline de integração e entrega contínua em uma instituição de ensino pública, contendo as etapas percorridas, ferramentas utilizadas, desafios superados, lições aprendidas e as impressões da equipe.

Palavras-chaves: integração contínua; entrega contínua; *time to market*; órgão público; economicidade.

ABSTRACT

In federal public education institutions, each year the student population grows, along with the number of employees, generating the need for new technological resources, such as, for example more efficient and higher quality information systems. Due the development teams are small, with little maturity in processes, there are delays in fulfill the demands, in addition to the build process and delivery of the software product being costly and not repeatable, caused by excessive errors, it delays delivey and implementation , resulting in frequent corrections, causing delivery to take hours or days without product quality assurance. This is due to the high number of errors that are reported late, causing delays and consequently missed deadlines. This work aimed to carry out a study on the adoption of integration and continuous delivery regarding the development software delivery cycle at Federal Institute of Acre (IFAC), in order to verify the impact of this adoption on the delivery process and product quality. To carry out this study, a systematic literature review (SRL) was performed that analyzed 54 studies published between 2016 and 2019, to collect evidence on challenges faced, best practices and tools used in the adoption and use of integration and continuous delivery. During the systematic review, 32 challenges, 20 best practices and 63 tools were found. From the evidence collected, a proposal for an evidence-based approach was built for the process of adoption and risk management of an integration and continuous delivery pipeline, through the relationship and combination of challenges and solutions (best practices and tools). As a validation of the approach, it was applied in an IFAC pilot project, being evaluated for feasibility, through the GQM, after adoption, using the method of comparison of the sister project. The main contributions of this work were: (1) the creation of a body of knowledge, from the evidence collected as a result of the systematic review, which works as a framework to compose the concrete and effective pipeline to be implemented; (2) a pipeline adoption process; and (3) an approach to risk management utilizing the body of knowledge. Finally, the process created in this research was applied in an IFAC pilot project and documented through an experience report presenting minimum strategies to adopt the integration and continuous delivery pipeline in a public education institution, containing the steps taken, tools used , challenges overcome, lessons learned and team feeling.

Keywords: continuous integration; continuous delivery; time to market; public agency; economics.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

APF	Administração Pública Federal
BDD	Desenvolvimento Guiado por Comportamento, em inglês *Behavior Driven Development*
BPMN	Modelo e Notação de Processos de Negócio, em inglês *Business Process Model and Notation*
CALMS	*Culture, Automation, Lean Measurement* e *Sharing*
CAMS	*Culture, Automation, Measurement* e *Sharing*
CD	Entrega Contínua, em inglês *Continuos Delivery*
CDE	Implantação Contínua, em inglês *Continuos Deployment*
CDN	Rede de Entrega de Conteúdo, em inglês *Content Delivery Network*
CGU	Controladoria Geral da União
CI	Integração Contínua, em inglês *Continuous Integration*
COGTI	Coordenação de Governança da Tecnologia da Informação
COSEG	Coordenação de Segurança da Informação
COSIN	Coordenação de Suporte e Infraestrutura
COSIS	Coordenação de Sistema de Informação
DevOps	Desenvolvimento e Operações, em inglês *Development and Operations*
DSGTI	Diretoria Sistêmica de Gestão de Tecnologia da Informação
GQM	*Goal Question Metric*
IF	Instituto Federal
IFAC	Instituto Federal do Acre
IFRN	Insituto Federal do Rio Grande do Norte
IN	Instrução Normativa
MASP	Método de Análise e Solução de Problemas
MMP	Produto Mínimo Vendável, em inglês *Minimal Marketable Product*

MVP	Mínimo Produto Viável, em inglês *Minimum Viable Product*
PDI	Plano de Desenvolvimento Institucional
PIT	Plano Individual de Trabalho
PROEJA	Programa Nacional de Integração da Educação Profissional com a Educação Básica de Jovens e Adultos
QA	*Quality Assurance*
RAD	Regulamentação das Atividades Docentes
RIT	Relatório Individual de Trabalho
RSL	Revisão Sistemática da Literatura
SDLC	Ciclo de Vida de Desenvolvimento de Software, em inglês *Software Development Life Cycle*
SFTP	*SSH File Transfer Protocol*
SIG	Sistema Integrado de Gestão
SISRAD	Sistema de Regulamentação de Atividades Docentes
SQL	*Structured Query Language*
SRE	*Site Reliability Engineering*
SUAP	Sistema Unificado de Administração Pública
TBD	Desenvolvimento Baseado no Trunk, em inglês *Trunk-based development*
TCU	Tribunal de Contas da União
TDD	Desenvolvimento Orientado a Teste, em inglês *Test Driven Development*
TI	Tecnologia da Informação
TIC	Tecnologia da Informação e Comunicação
UFRN	Universidade Federal do Rio Grande do Norte
VCS	Sistema de Controle de Versão, em inglês *Version Control Systems*
VM	Máquina Virtual, em inglês*Virtual Machine*
XP	*Extreme Programming*

SUMÁRIO

1 INTRODUÇÃO

A crescente demanda dos usuários por novos serviços intensivos em Tecnologia da Informação e Comunicação (TIC) tem exigido dos desenvolvedores um aumento na qualidade dos produtos entregues e com prazos de entrega cada vez menores, o que levou a um mercado competitivo, trazendo mudanças no ciclo de desenvolvimento e de entrega (BERNARDO; COSTA; KULESZA, 2018).

Existem diversas iniciativas na literatura que são amplamente adotadas na industria de software. Elas são utilizadas para entregar produtos de alto valor no mercado, de maneira rápida e alinhada com as novas tendências, como é o caso do DevOps e do *Site Reliability Engineering* (SRE).

Segundo Beyer et al. (2018), o DevOps é um conjunto solto de práticas, diretrizes e cultura projetado para quebrar silos no desenvolvimento de Tecnologia da Informação (TI), operações, rede e segurança. Ele está centrado no objetivo de encorajar uma maior contribuição entre os envolvidos na entrega do produto, gerando um maior valor para o cliente, com confiança e rapidez (CRUZ; ALBUQUERQUE, 2018), o que causa impactos diretos no crescimento das empresas e na qualidade dos sistemas desenvolvidos (RIUNGU-KALLIOSAARI et al., 2016). Dentre as práticas da metodologia DevOps, destacam-se a Integração Contínua (CI) e a Entrega Contínua (CD).

A Integração Contínua (CI), incentiva o desenvolvedor a integrar seu trabalho com frequência, sendo que, cada integração é verificada por um *build* automatizado que inclui a execução de testes, sabendo que, se forem detectados erros de integração, receberá *feedback* imediato, dando tempo para corrigi-los o mais rápido possível (DUVALL; MATYAS; GLOVER, 2007; FOWLER, 2006).

A Entrega Contínua (CD), como uma extensão da CI, defende a automação dos testes, incorporando processos adicionais para garantir que o código que foi integrado esteja apto para a implantação (CRUZ; ALBUQUERQUE, 2018).

Com a utilização da Integração e da Entrega Contínua espera-se vários benefícios, tais como: (1) obter *feedback* maior e mais rápido dos clientes e do processo de desenvolvimento de software; (2) levar à melhoria da satisfação do cliente e da qualidade do produto, com lançamentos frequentes e confiáveis; (3) redução do tempo de lançamento do produto.

O conceito de SRE foi criado por Ben Treynor, em 2003, no Google. O SRE é uma

mentalidade e um conjunto de práticas, métricas e formas prescritivas para garantir a confiabilidade dos sistemas, funcionando como uma implementação específica do DevOps com algumas extensões (BEYER et al., 2016). Dentre benefícios de adotar o SRE, destacam-se: (1) gerenciamento de incidentes; (2) otimização do produto desde o desenvolvimento; (3) cumprimento de SLA com menos esforços; (4) agilidade na entrega de serviço; (5) e aumento da eficiência operacional.

De acordo com Beyer et al. (2018), DevOps e SRE estão, tanto na prática quanto na filosofia, muito próximos um do outro no cenário geral das operações de TI. No entanto, elas se diferenciam principalmente com relação ao foco. No SRE, o foco é melhorar a disponibilidade e confiabilidade do sistema, já o DevOps se concentra na velocidade de desenvolvimento e entrega, ao mesmo tempo em que reforça a continuidade.

No entanto, a agilidade com que o mercado entrega soluções de TI, muitas vezes não é encontrada no governo, que não consegue acompanhar o ritmo de evolução das tecnologias e metodologias de desenvolvimento, tendo como resultado os desenvolvimentos lentos, caros e que não atendem as demandas dos cidadãos (GERMANI et al., 2016).

Os Institutos Federais de Educação Ciência e Tecnologia (Institutos Federais) fazem parte da Rede Federal de Educação Profissional, Científica e Tecnológica, criada por meio da Lei n° 11.892, de 29 de dezembro de 2008[1], com o objetivo de ampliação, interiorização e diversificação da educação profissional e tecnológica no país. Desde sua criação, os Institutos Federais (IF) vêm passando por expansões, atualmente são compostos por 38 Instituições, presentes em todos os estados e no Distrito Federal. Nos IF, os projetos de desenvolvimento de software são geralmente de pequeno e médio porte, visando a solução de problemas pontuais, a integração entre os sistemas, manutenções e aprimoramentos dos mesmos.

A cada ano a população estudantil cresce, juntamente com o número de docentes e técnicos administrativos, gerando a necessidade de novos recursos tecnológicos, como sistemas de informação mais eficientes e com maior qualidade, que visam uma maior eficiência na utilização de recursos públicos, desburocratização, melhoria dos processos administrativos, melhoria da satisfação dos usuários. Embora a cultura DevOps tenha se difundido nas grandes empresas de desenvolvimento privadas, nos órgãos públicos brasileiros essa prática ainda não é tão aplicada, devido a eles serem considerados instituições burocráticas (SIQUEIRA et al., 2018).

[1] Disponível em: <https://www.planalto.gov.br/ccivil_03/_ato2007-2010/2008/lei/l11892.htm>. Acesso em: 23 out. 2020

1.1 MOTIVAÇÃO

As organizações públicas possuem cada vez mais necessidades que podem e devem ser atendidas pelos sistemas de computador. Isto se deve ao fato de a forma de gerir os serviços públicos ter sido modificada, o que colocou o cidadão na posição de cliente, com a utilização do governo eletrônico ou *e-gov* (BARBOZA, 2019, apud Abel, 2017), que consiste no uso de TIC na entrega dos produtos e serviços do Estado a população. Além de esses serviços estarem cada vez mais integrados.

Em razão de as instituições públicas possuírem características que diferem da iniciativa privada, torna-se difícil o uso de metodologias ágeis no desenvolvimento de software, pois a maior parte delas não objetivam lucro, em vez disso, sua finalidade se resume em gerar produtos e serviços demandados pela sociedade. Outra característica dessas instituições, refere-se ao processo de gestão, que por possuírem complexas hierarquias políticas na cadeia de comando, este processo pode sofrer influencia nos seus diversos níveis.

Barboza (2019) elencou diversas dificuldades em utilizar metodologia ágil no setor público, dentre elas destacam-se a cultura organizacional, burocracia, o modo como os servidores estão habituados a trabalhar dificuldades e inovações utilizando ferramentas modernas como as eletrônicas de modo arcaico.

Silva, Carvalho e Santos (2012) dizem que as equipes de desenvolvimento pequenas e com pouca maturidade em processos são as que mais sofrem para quebrar o ciclo ao qual estão submetidas, visto que, estão trabalhando constantemente de forma reativa, desperdiçando um grande tempo em retrabalhos.

As equipes de desenvolvimento dos IF são normalmente pequenas e com pouca maturidade em processos, elas têm sua força de trabalho reduzida devido a realizarem tarefas de suporte e implantação de sistemas de terceiros, além de outras tarefas administrativas e as relacionadas ao desenvolvimento, podendo acarretar atrasos no projeto e dificultando a quebra do ciclo ao qual estão submetidas, visto que, estão trabalhando constantemente de forma reativa, desperdiçando recursos e um grande tempo em retrabalhos (SILVA; CARVALHO; SANTOS, 2012), ferindo o princípio da economicidade.

O processo de *build* e entrega do produto de software é custoso e não repetível, devido ao excesso de erros, demora para fazer a entrega e a implantação, acarretando em correções frequentes, fazendo com que a entrega demore horas ou dias sem garantia da qualidade do produto, pois um alto número de erros que são reportados tardiamente, gerando atrasos e

consequentemente a perca dos prazos.

A economicidade é um dos princípios da administração pública expressamente previsto no Art. 70 da Constituição Federal de 1988[2] e representa, em síntese, a prestação do serviço ou no trato com os bens públicos com qualidade, celeridade e menor custo possível.

Visando a economicidade pública, um estudo voltado para análise do processo de entrega e da qualidade dos produtos de software produzidos por um Instituto Federal, considerando o uso de práticas contínuas, ensejando propor uma abordagem que apoie a adoção de tais práticas pro outros institutos, mostra-se como uma boa oportunidade de pesquisa.

1.2 QUESTÃO DE PESQUISA

Os Institutos Federais de Educação têm se tornado referência com relação a educação profissional pública, gratuita e de qualidade, buscando sempre prover aos menos favorecidos a oportunidade de ingressar em uma instituição federal de ensino. Apesar das dificuldades em utilizar práticas contínuas no setor público, surge a seguinte questão de pesquisa: "Como integração e a entrega contínua podem ser utilizadas no IFAC afim de reduzir o tempo de inserção no mercado (*time to market*) e o número de erros dos novos recursos do produto de software?".

1.3 HIPÓTESES

A partir da questão de pesquisa, é possível definir as seguintes hipóteses a serem comprovadas pelo presente trabalho: H1) adotar integração e entrega contínua no processo de desenvolvimento e entrega do produtos de software dos IF reduz o tempo em retrabalhos e aumenta a qualidade do produto; H2) o comprometimento dos envolvidos com a adoção de um pipeline de integração e entrega contínua traz benefícios para a equipe e para o projeto.

1.4 OBJETIVOS

Este trabalho teve como **objetivo geral** realizar um estudo sobre a adoção da integração e entrega contínua quanto ao ciclo de entrega do desenvolvimento de software na DSGTI do

[2] Disponível em: <https://www.planalto.gov.br/ccivil_03/Constituicao/Constituicao.htm>. Acesso em: 10 nov. 2020.

IFAC, de forma a verificar o impacto dessa adoção no processo de entrega e qualidade do produto.

Quanto aos **objetivos específicos** abaixo listados, complementam o objetivo geral proposto:

- Propor um processo de adoção de pipeline de integração e entrega contínua utilizando o corpo de conhecimento;
- Propor uma abordagem para o gerenciamento de risco do pipeline de integração e entrega contínua utilizando o corpo de conhecimento;
- Apresentar estratégias mínimas para adotar o pipeline de integração e entrega contínua utilizando a abordagem proposta, através do relato de experiência da adoção e uso dessas práticas em um projeto piloto do IFAC;

1.5 ORGANIZAÇÃO DO TRABALHO

Além desse capítulo, esta dissertação encontra-se estruturada da seguinte forma.

Capítulo 2 - Referencial Teórico: é apresentado o referencial teórico, que contém a revisão das teorias bases da dissertação. Inicialmente é explicitado a diferença entre implantação e *release* de *software*. Em seguida é apresentado o DevOps e os principais conceitos. Por fim, são apresentados os conceitos, práticas e princípios relacionados a Integração e Entrega Contínua.

Capítulo 3 - Trabalhos Correlatos: são apresentados os trabalhos correlatos a esta pesquisa cujo o foco seja a analise dos fatores de sucesso envolvidos na implantação de integração e entrega contínua, descrevendo o objeto de estudo e as principais contribuições. Por fim, é feita uma análise comparativa entre os trabalhos correlatos a esta pesquisa, apresentando as lacunas existentes e identificando que esta pesquisa se difere na questão do objeto de estudo.

Capítulo 4 - Metodologia de Pesquisa: descreve a metodologia empregada para realizar o estudo, a classificação da pesquisa junto ao quadro metodológico, as principais etapas da pesquisa, o processo para realização da revisão sistemática da literatura (RSL), o método utilizado para avaliar a abordagem produzida com base nas evidências coletadas da . Por fim, é descrita a forma de análise dos dados extraídos na RSL.

Capítulo 5 - Execução e Resultados: são apresentados os resultados obtidos da revisão sistemática. Inicialmente o capítulo apresenta uma análise dos dados gerais da revisão sistemática como as principais fontes, o número de estudos retornados, a distribuição temporal dos estudos, a avaliação da qualidade dos mesmos, entre outras informações. Em seguida são apresentadas as evidências para as questões de pesquisa. Por fim, é realizada uma análise dos resultados obtidos.

Capítulo 6 - Estratégia para Adotar Integração e Entrega contínua: descreve o relato de experiência da adoção da integração e entrega contínua em projeto piloto do IFAC utilizando a abordagem construída. Por fim, é mostrada a avaliação da abordagem utilizando o método de comparação do projeto irmão.

Capítulo 7 - Conclusão: apresenta as considerações finais sobre os principais tópicos abordados nesta dissertação, incluindo as contribuições alcançadas e as indicações de trabalhos futuros.

Por fim, temos os Apêndices A, B, C, D, E e F, os quais são materiais complementares que se mostram necessários para a compreensão da dissertação.

2 REFERENCIAL TEÓRICO

A base teórica necessária para a realização da pesquisa e o entendimento do estudo é apresentada nesse capítulo. O capítulo está organizado da seguinte forma:

1. **Entrega de Software** (seção 2.1): esta seção apresenta os principais conceitos entrega de software, diferenciando implantação e *release*.
2. **DevOps** (seção 2.2): esta seção apresenta a definição do termo DevOps, seus princípios, práticas e ciclo de vida.
3. **Integração Contínua** (seção 2.3): esta seção conceitua a integração contínua além de apresentar princípios, práticas e os benefícios que essa prática traz.
4. **Entrega Contínua** (seção 2.4): esta seção conceitua a entrega contínua, apresentando princípios, práticas e os benefícios que essa prática traz, além de diferenciá-la da prática de implantação contínua.

2.1 ENTREGA DE SOFTWARE

A cada dia, precisamos implantar com mais frequência para alcançar o resultado desejado de fluxo suave e rápido, não com menos frequência. Para habilitar isso, se faz necessário diferenciar os conceitos de implantação em produção dos lançamentos (*release*) de recursos. Na prática, os termos implantação e *release* são frequentemente usados de forma intercambiável. No entanto, segundo Kim et al. (2016) são duas ações distintas que servem a dois propósitos muito diferentes:

- A implantação é a instalação de uma versão especificada do software em um determinado ambiente (por exemplo, implantação de código em um ambiente de teste de integração ou implantação de código em produção). Especificamente, uma implantação pode ou não estar associada ao lançamento de um recurso para os clientes.
- Liberação (*release*) é quando disponibilizamos um recurso (ou conjunto de recursos) para todos os nossos clientes ou um segmento de clientes (por exemplo, permitimos que o recurso seja usado por 5% de nossa base de clientes).

Para Visser et al. (2017) é necessário criar ambientes separados para vários estágios do Ciclo de Vida de Desenvolvimento de Software, em inglês *Software Development Life Cycle* (SDLC). Esses ambientes devem ser o mais semelhantes possível e possuir um controle rigoroso da progressão do código de um ambiente para outro, o que levará a uma melhora na velocidade e a previsibilidade do desenvolvimento, pois os defeitos encontrados em cada ambiente podem ser diagnosticados de forma clara e corrigida sem atrapalhar o fluxo de desenvolvimento. Esses autores defendem que devem existir quatro ambientes: Desenvolvimento, Teste, Aceitação e Produção.

Quando existe um encadeamento destes ambientes, com conceitos de *build*, *release* e *deploy* associados, pode-se afirmar que se está perante um pipeline de desenvolvimento de software que suporta o SDLC (VIRMANI, 2015).

No âmbito do DevOps, o pipeline é automatizado, surgindo os conceitos de Integração Contínua, em inglês *Continuous Integration* (CI), Entrega Contínua, em inglês *Continuous Delivery* (CD) e Implantação Contínua, em inglês *Continuous Deployment* (CDE), que propõem preencher as lacunas existentes nas metodologias de processo prescritivos e ágeis, visando ajudar as organizações a acelerar seu desenvolvimento e entrega de recursos de software sem comprometer a qualidade. (COIS; YANKEL; CONNELL, 2014; SHAHIN; ALI BABAR; ZHU, 2017).

2.2 DEVOPS

DevOps é um acrônimo das palavras Desenvolvimento (*Development*) e Operações (*Operations*), que descreve um conjuntos de práticas que pretendem reduzir o tempo entre o *commit* da alteração de um sistema e essa mudança ser disponibilizada em produção, garantindo a alta qualidade (BASS; WEBER; ZHU, 2015).

O termo "surgiu" inicialmente no evento *O'Reilly Velocity Conference* em 2009, onde John Allspaw e Paul Hammond apresentaram a palestra de título: *"10+ Deploys Per Day: Dev and Ops Cooperation at Flickr"* que contava sobre os resultados e desafios da maior aproximação entre a equipe de desenvolvimento e de operações no Flickr. Patrick Debois, que já tentava resolver o problema da divisão entre desenvolvedores e pessoal de operações, após assistir a palestra, decidiu então, organizar um envento em Ghent na Bélgica, chamado *DevOpsDays*, o qual houve muito interesse e a conferência foi um sucesso. O "DevOpsDay" se tornou um evento recorrente e em conversas no Twitter e em vários fóruns da Internet foi abreviado para "DevOps" (VERONA, 2016).

A cultura DevOps é considerada uma extensão das metodologias ágeis para as etapas de desenvolvimento e operação de sistemas, sendo uma maneira de gerenciar o ciclo de entrega do produto de ponta a ponta (IBRAHIM; SYED-MOHAMAD; HUSIN, 2019). Tem como objetivos loops de feedbak curtos, automação, versões frequentes, colaboração no ciclo de desenvolvimento entre as equipes de desenvolvimento e de operações (LWAKATARE et al., 2016).

O DevOps se baseia em culturas de gerenciamento de alta confiança, liderança servidora e gerenciamento de mudanças organizacionais. O resultado é qualidade, confiabilidade, estabilidade e segurança de classe mundial com custo e esforço cada vez menores; e fluxo e confiabilidade acelerados em todo o fluxo de valor de tecnologia, incluindo gerenciamento de produtos, desenvolvimento, controle de qualidade, operações de TI e infosec. (KIM et al., 2016)

2.2.1 Pilares do DevOps

Willis (2010) propôs que o DevOps poderia ser caracterizado em quatro pilares: cultura (*culture*), automação (*automation*), medição (*measurement*) e compartilhamento (*sharing*). A siga CAMS originou-se dessas quatro expressões (HUMBLE; MOLESKY, 2011).

Cada letra do CALMS representa uma questão a ser trabalhada na organização, a fim de implementar uma verdadeira cultura DevOps. Abaixo são apresentados os cinco pilares do DevOps, simbolizados pela sigla CALMS:

- **Cultura (*Culture*)**: A implementação do DevOps traz consigo a proposta de uma nova cultura e mentalidade organizacional (HUMBLE; MOLESKY, 2011; RILEY, 2014). A cultura é necessária para remover barreiras e facilitar a colaboração para atingir os objetivos da equipe.

- **Automação (*Automation*)**: DevOps depende da automação total da construção, implantação e teste para atingir prazos de entrega curtos e, consequentemente, entrega rápida e *feedback* dos usuários finais (FITZGERALD; STOL, 2017), portanto deve-se liberar os humanos para realizar tarefas que exigem criatividade e intuição e deixar as tarefas repetitivas para os computadores, que sabem executá-las rapidamente e de forma bem mais confiável.

- **Pensamento Enxuto (*Lean*)**: diversos princípios da metodologia *lean* influenciam a cultura DevOps, como o foco no cliente e no valor gerado para ele, melhoria contínua, o

foco em qualidade, a eliminação de desperdícios, a otimização, e o respeito às pessoas.

- **Medição (*Measurement*)**: Obter uma compreensão da capacidade de entrega atual e definir metas para melhorá-la só pode ser feito por meio de medição (FITZGERALD; STOL, 2017). Uma adoção de DevOps de sucesso irá medir tudo o que for possível, por exemplo: métricas de negócios, métricas de desempenho, cobertura de testes, o tempo para implantar uma nova versão, etc.

- **Colaboração (*Sharing*)**: a colaboração e o compartilhamento de ideias e conhecimento ajudam a criar a cultura necessária para o sucesso com DevOps. Com o compartilhamento de informações, caso uma pessoa saia do processo, isso não interferirá no andamento do projeto, nem interferirá no funcionamento do fluxo de desenvolvimento, tornando-o autossustentável.

2.2.2 Princípios do DevOps

Segundo Kim et al. (2016), "*The Three Ways*", são os princípios dos quais todos os padrões de DevOps podem ser derivados. Os princípios do "*Three Ways*" descrevem os valores e filosofias que enquadram os processos, procedimentos, práticas de DevOps, bem como as etapas prescritivas. O *Three Ways* é divido em três categorias: Fluxo, Feedback e Aprendizagem e Experimentação Contínua.

Os princípios do Fluxo (*The First Way*) aceleram a entrega do trabalho do Desenvolvimento às Operações para nossos clientes, permitindo um fluxo de trabalho rápido da esquerda para a direita, do desenvolvimento às operações para o cliente. Para maximizar o fluxo, precisamos tornar o trabalho visível, reduzir nossos *batch sizes* (em português tamanhos de lote), ou seja, a quantidade de trabalho que é feito antes de liberar ou integrar, e intervalos de trabalho, aumentar a qualidade evitando que os defeitos sejam passados para as etapas do fluxo de trabalho posteriores e otimizar constantemente para as metas globais.

Ao acelerar o fluxo, reduz-se o tempo de espera necessário para atender às solicitações internas ou do cliente, especialmente o tempo necessário para implantar o código no ambiente de produção. Ao fazer isso, aumentamos a qualidade do trabalho, bem como nosso rendimento, e aumentamos nossa capacidade de superar a concorrência.

Ao colocarmos esse princípio em prática obtemos como resultados nunca passar um defeito conhecido para etapas posteriores, nunca permitir que a otimização local crie degradação

global, sempre buscar aumentar o fluxo e sempre buscar alcançar uma compreensão profunda do sistema.

Os princípios do *Feedback (The Second Way)* permitem o fluxo rápido e constante de *feedback* da direita para a esquerda em todos os estágios do nosso fluxo de valor. Requer que seja amplificado o *feedback* para evitar que problemas voltem a acontecer ou permitir uma detecção e recuperação mais rápidas. Ao fazer isso, cria-se qualidade na fonte e é gerado ou é incorporamos conhecimento onde é necessário, permitindo criar sistemas de trabalho cada vez mais seguros onde os problemas são encontrados e corrigidos muito antes de ocorrer uma falha catastrófica.

Como resultados da *The Second Way* pode-se entender e responder a todos os clientes, internos e externos, encurtando e ampliando todos os ciclos de *feedback* e incorporando conhecimento onde precisamos dele.

Os princípios de Aprendizagem e Experimentação Contínua (*The Third Way*) trata da criação de uma cultura que promova duas coisas: experimentação contínua, assumir riscos e aprender com o fracasso; e compreender que a repetição e a prática são pré-requisitos para a maestria. De forma que, ao encurtar e ampliar continuamente nossos ciclos de *feedback*, são criados sistemas de trabalho cada vez mais seguros, nos quais a equipe se torna mais capazes de assumir riscos e realizar experimentos que os ajudam a aprender mais rápido do que os concorrentes e vencer no mercado.

Os resultados da *The Third Way* incluem alocar tempo para a melhoria do trabalho diário, criar rituais que recompensam a equipe por assumir riscos e introduzir falhas no sistema para aumentar a resiliência.

Além dos princípios norteadores do DevOps, Kim et al. (2016) elencam princípios básicos que devem ser seguidos de forma a que a curva de adoção do DevOps seja o mais linear possível:

- Testar sempre que for possível, de modo a garantir a integridade de todo o ciclo de desenvolvimento e da própria aplicação;
- Melhoria contínua dos processos implementados, com a adoção de novas técnicas e ferramentas que visem criar uma maior qualidade do produto para que as entregas sejam ainda mais rápidas;

- Automatizar tudo o que for possível, para evitar erro humano e investimento desnecessário em recursos humanos;
- Trabalhar de forma a criar uma única equipa para trabalhar em torno dos mesmos objetivos, eliminando assim os silos existentes;
- Separar o processo por áreas e atividades, mas não de forma disruptiva que vá contra o princípio anterior, mas de forma a que se saiba onde começa e termina cada uma das fases do ciclo.

Por fim, deve-se estar sempre ter o conhecimento das novas tecnologias de forma a que se consiga criar um Mínimo Produto Viável, em inglês *Minimum Viable Product* (MVP), com novas soluções que melhorem o modelo de negócio.

2.2.3 Práticas do DevOps

Há diversas práticas de DevOps, por exemplo, na execução de testes, implantação e monitoramento, a escolha das práticas que devem ser implementadas depende do caso de uso específico. Dentre as práticas DevOps destacam-se das de automação que podem ser agrupadas nas categorias de Integração Contínua (CI), de Entrega Contínua (CD), de Implantação Contínua (CDE) e de Monitoramento (IVANOV; SMOLANDER, 2018; SHAHIN; ALI BABAR; ZHU, 2017).

- A Integração Contínua (CI) pode ser definida como uma prática de desenvolvimento de software em que os desenvolvedores mesclam suas alterações de código na base de código compartilhada com a maior frequência possível e essas alterações são validadas executando ferramentas de qualidade de código, construindo o projeto e executando testes automatizados;
- A Entrega Contínua (CD) é o próximo passo após o CI que garante que um projeto esteja pronto para ser lançado a qualquer momento mediante solicitação. Ele pode ser liberado para ambientes de desenvolvimento, preparação ou produção. A Implantação Contínua (CDE) é outro termo semelhante, o que significa que a implantação de compilações bem-sucedidas no ambiente escolhido acontece automaticamente após cada confirmação em ramificações selecionadas;

- O Monitoramento Contínuo é o conjunto de práticas usadas para monitorar o comportamento de tempo de execução dos aplicativos para detecção precoce de problemas, como degradação de desempenho ou erros de lógica de negócios.

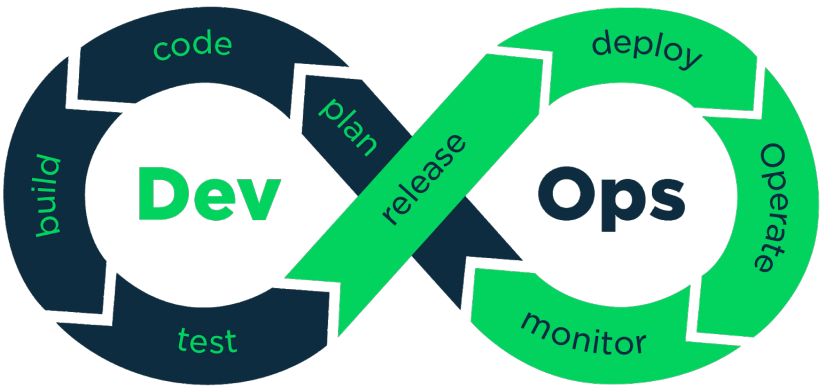
De acordo com Ivanov e Smolander (2018), as práticas de CI e CD juntas podem ser chamadas de *pipeline* de engenharia de lançamento. Esse *pipeline* também pode ser chamado com um termo mais amplo, como *pipeline* DevOps, especialmente se incluir práticas adicionais, como Monitoramento Contínuo.

Portanto, pode-se afirmar que o uso do CI/CD repercute na produção de novas versões de software (*releases*) mais rapidamente e com mais frequência, de forma a eliminar os obstáculos que existem entre o desenvolvimento e a produção, fazendo com que o cliente seja beneficiado, pois obtém um acesso mais rápido e frequente a novas funcionalidades.

2.2.4 Ciclo de Vida do DevOps

O ciclo de DevOps apresentado na Figura 1 ilustra todos os processos inerentes ao desenvolvimento de uma aplicação.

Figura 1 – Ciclo DevOps



Fonte: Auctus (2017)[1]

O ciclo de DevOps contem 8 etapas que formam um ciclo continuo, sendo que cada etapa possui funções bem estabelecidas.

A primeira etapa é o planejamento, ou seja, antes que a equipe comece os seus trabalhos, é preciso planejar cada uma das atividades, estabelecer metas e padrões de qualidade, explicar

[1] Disponível em: <https://www.auctus.com.br/produto/devops/>. Acesso em: 21 abril. 2021.

processos e criar o escopo do projeto. Com as definições em mãos os desenvolvedores estarão aptos a transformar os objetivos em código.

A próxima etapa é o codificação, aqui os desenvolvedores estão fazendo o desenvolvimento e a revisão do código. Nesta etapa se faz necessário o uso de uma ferramenta de versionamento de código, onde o trabalho será dividido em pequenas partes para uma produção melhor e mais rápida, ou seja, o desenvolvedor deve *commitar* o código em pequenos pedaços várias vezes ao dia para tornar o teste mais simples e fácil.

Na etapa de *build*, a cada alteração no repositório de código central, o código-fonte será construído e serão executados alguns testes iniciais, como os testes unitários, de forma automática em um formato desejado que permita ser testado e implantado.

Na etapa de teste serão executados os testes de aceitação, integridade, performance e não funcionais entre outros, de forma a permitir que os problemas possam ser identificados e resolvidos assim que ocorrem, evitando que isso gere atrasos e transtornos para o projeto como um todo.

As etapas de *release* e de *deploy* estão diretamente relacionadas, pois enquanto a de *release* sinaliza que o código está pronto para ser implantado, a de *deploy* é a execução do processo de implantação no ambiente de produção.

As duas etapas finais são de operações e monitoramento, onde as equipes garantem a integridade e manutenção das aplicações, recolhendo métricas para os próximos desenvolvimentos de forma a assegurar uma melhoria constante, além de permitir corrigir incidentes com mais rapidez e criar uma melhor experiência para o seu usuário final.

Desta forma, através deste ciclo e com *feedback* constante em cada fase é possível entregar software ao cliente final com mais valor.

2.3 INTEGRAÇÃO CONTÍNUA

A Integração Contínua, em inglês *Continuous Integration* (CI), tem origem no processo de desenvolvimento de *Extreme Programming* (XP) de Kent Beck, como uma de suas doze práticas originais, que defendia a necessidade de entregar e testar pequenas mudanças reduzindo o risco de erros e facilitando a depuração (FOWLER, 2006). Embora o CI seja uma prática que não requer ferramentas específicas para implantação, descobrimos que é útil usar um servidor de Integração Contínua.

Integração continua é um conjunto de práticas de desenvolvimento de software, no qual

os desenvolvedores integram seus códigos diariamente, e essa integração é verificada por uma compilação automática do código, a qual são executados os teste unitários e de integração, é validada a cobertura do código e verificada a conformidade com os padrões de codificação para que a detecção de erros de integração seja mais rápida (HUMBLE; FARLEY, 2014; FOWLER, 2006). Já para Duvall, Matyas e Glover (2007), o CI não é apenas uma implementação técnica, mas é também uma implementação organizacional e cultural, pois as pessoas geralmente resistem à mudança, e a melhor abordagem para uma organização pode ser adicionar os mecanismos automatizados ao processo.

Embora haja alguma forma de automação do processo, a frequência da integração também é importante, pois deve ser regular o suficiente para garantir um *feedback* rápido aos desenvolvedores. Por fim, falhas de integração contínua são eventos importantes que podem ter uma série de cerimônias e artefatos altamente visíveis para ajudar a garantir que os problemas que levam a essas falhas sejam priorizados para solução o mais rápido possível por quem for considerado responsável.(FITZGERALD; STOL, 2017)

A pratica da integração contínua acontece toda vez que uma nova atualização do sistema é enviada para o repositório de código. Primeiramente, o desenvolvedor A implementa a nova funcionalidade em uma cópia local do repositório, terminada a implementação ele faz um *commit* local, executa localmente o *script* de *build* e os testes automáticos, caso execute tudo com sucesso ele poderá pensar em enviar as alterações para o repositório central.

Caso haja mudanças na ramificação principal do repositório central realizadas por outros desenvolvedores, antes de enviar as alterações, o desenvolvedor A deve atualizar sua cópia de trabalho com as alterações do repositório central e executar novamente o *script* de *build* e os testes automáticos localmente.

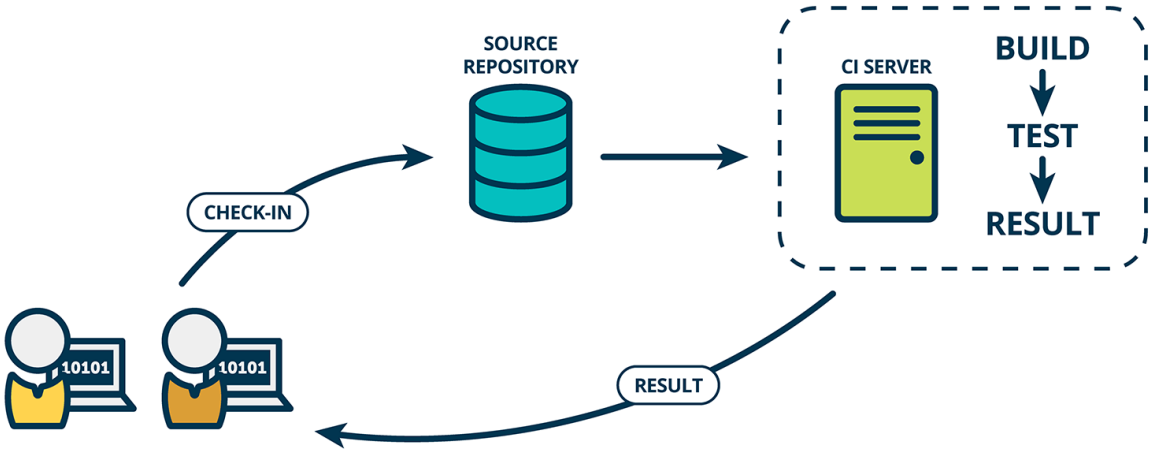
Se as alterações dos outros desenvolvedores conflitarem com as dele, isso se manifestará como uma falha na compilação ou nos testes. Neste caso, é responsabilidade do desenvolvedor A corrigir isso e repetir até que possa ser executado o *build* da cópia de trabalho local com sucesso e esta esteja devidamente sincronizada com a ramificação principal.

Uma vez que o desenvolvedor A tenha executado sua própria compilação com sucesso de uma cópia de trabalho devidamente sincronizada, ele pode finalmente confirmar suas alterações na ramificação principal do repositório central.

O servidor CI detectará as mudança no repositório central, pega uma cópia do código e executa o *script* de *build* e os testes automáticos, com a finalidade de cobrir todo o código, inclusive as partes recém adicionadas.

Caso haja algum problema durante a execução, o servidor CI notificará a equipe de desenvolvimento de que o *build* não foi realizado. A responsabilidade da correção da falha do processo de compilação é do desenvolvedor cujo o *commit* causou a falha, ou seja, somente quando o processo de *build* no servidor CI for concluída com sucesso o trabalho do desenvolvedor estará finalizado. Na Figura 2 é possível visualizar o processo de integração contínua.

Figura 2 – Diagrama da Integração Contínua



Fonte: Deploybot Blog (2018)[2]

Quando houver falha de compilação reportada pelo servidor de integração, ela deve ser corrigida rapidamente fazendo com que o processo de compilação volte a funcionar corretamente, ou seja, não se deve ter uma falha de compilação de integração por muito tempo. Segundo Fowler (2006), uma boa equipe deve ter muitas construções corretas por dia.

O resultado de se adotar o CI é que tem-se um software estável que funciona corretamente e contém poucos *bugs*, isto é, todas as novas funcionalidades serão desenvolvidas a partir de uma base estável, que é repositório central, e os desenvolvedores nunca devem ficar muito distante e/ou demorar muito para se integrar a ela. Portanto, menos tempo é gasto tentando encontrar *bugs* porque eles aparecem rapidamente.

2.3.1 Práticas e Princípios da Integração Contínua

Existem algumas práticas necessárias para que a Integração Contínua funcione corretamente, as quais devem ser seguidas pelos desenvolvedores. Duvall, Matyas e Glover (2007)

[2] Disponível em: <https://deploybot.com/blog/continuous-development>. Acesso em: 10 dez. 2020.

estabelecem as seguintes práticas:

- ***Commit* de código frequentemente**: essa prática facilita a integração e permite que os desenvolvedores possam usar as alterações mais recentes. Para isso, deve-se fazer pequenas alterações sem alterar muitos componentes de uma vez, ou seja, escolher uma pequena tarefa e ao conclui-la, realize o commit e envia-a ao repositório central;

- **Não fazer *commit* de código quebrado**: a boa prática nesse caso é executar o *build* e o testes no ambiente de desenvolvimento antes de fazer o commit as alterações e envia-las ao repositório central;

- **Corrija compilações quebradas imediatamente**: toda vez que o processo de *build* apresentar algum problema como erro de compilação, um teste ou inspeção com falha, um problema com o banco de dados ou uma implantação com falha, a equipe deve ter como prioridade consertar a compilação quebrada. Conforme Fowler (2006) não significa que todos na equipe tenha que se envolver para consertar a compilação, geralmente são necessárias apenas algumas pessoas para resolve-la. Não é uma coisa ruim para o processo de compilação quebrar, no entanto, quando quebrar é importante que ele seja corrigido rapidamente;

- **Escrever testes automatizados**: para que um servidor CI executar os testes, os testes devem ser automatizados;

- **Todos os testes e inspeções devem passar**: para que o processo de *build* seja aprovado todos os testes automatizados de um projeto devem passar;

- **Executar *builds* privadas**: para evitar processos de *build* com falhas o desenvolvedor deve realizar simular a integração local depois de executar os testes unitários. Localmente o desenvolvedor obterá as alterações de outros desenvolvedores, obtendo as alterações mais recentes do repositório e execute uma construção de integração completa localmente, como forma de antecipar possíveis problemas, tornando mais improvável que a integração no repositório não falhe;

- **Evitar integrar novos códigos a códigos com *build* quebrado**: se a última alteração enviada para o repositório tiver causado uma falha no processo de *build* é melhor esperar a correção ou ajudar a corrigir a falha, caso contrário gastará o tempo desenvolvendo

uma solução alternativa para o erro conhecido que causou a falha, apenas para compilar e testar o código.

2.3.2 Benefícios da Integração Contínua

A adoção das práticas do CI fornecem diversos benefícios, como: (1) obter mais e rápido *feedback* do processo de desenvolvimento de software e dos clientes; (2) ter ciclos de lançamento mais curtos e frequentes e previsível, melhorando a qualidade do software e aumentando a produtividade de suas equipes (FITZGERALD; STOL, 2017; SHAHIN; ALI BABAR; ZHU, 2017).

Para Rodríguez et al. (2017), a principal vantagem do CI é que ele automatiza tarefas como compilação de código, execução de testes unitários e de aceitação, monitoramento e validação de cobertura de código, verificação de conformidade com padrões de codificação, análise estática de código, revisão automática de código e construção de pacotes de implantação. Portanto, o CI fornece mecanismos para garantir que haja sempre um produto despachável que tenha passado em todas as fases de teste, informando que a frequência da integração é tão importante quanto a própria automação. Assim, a frequência deve ser alta o suficiente para garantir um *feedback* rápido aos desenvolvedores.

De acordo com Duvall, Matyas e Glover (2007), o valor de CI é:

- **Redução de riscos:** ao integrar várias vezes ao dia, pode-se reduzir os riscos no projeto. Isso facilita a detecção de defeitos, a medição da integridade do software e a redução de suposições. Os defeitos são detectados e corrigidos mais cedo, isto é, como o CI integra e executa testes e inspeciona várias vezes ao dia, há uma chance maior de que os defeitos sejam descobertos quando são introduzidos em vez de durante testes de ciclo tardio;

- **Redução de processos manuais repetitivos:** com a redução dos processos repetitivos economiza-se tempo, custos e esforço. Ao automatizar o CI, obtém-se uma capacidade maior de garantir que o processo é executado sempre da mesma maneira; que um processo ordenado é seguido. Por exemplo, você pode executar inspeções (análise estática) antes de executar testes nos *scripts* de compilação; e que Os processos serão executados sempre que ocorrer uma confirmação no repositório de controle de versão. Isso facilita a redução do trabalho em processos repetitivos, liberando as pessoas para fazer um trabalho mais instigante e de maior valor;

- **Gerar software implementável a qualquer hora e em qualquer lugar**: com o CI pode-se permitir que haja liberação de software implantável a qualquer momento. Com CI, são feitas pequenas alterações no código-fonte e integra-se essas alterações com o restante da base de código regularmente. Se houver algum problema, os membros do projeto são informados e as correções são aplicadas no software imediatamente. Projetos que não adotam essa prática podem esperar até imediatamente antes da entrega para integrar e testar o software, podendo atrasar um lançamento, atrasar ou impedir a correção de certos defeitos, causando novos defeitos à medida que os desenvolvedores se apressam para concluir e podendo significar o fim do projeto;

- **Permitir uma melhor visibilidade do projeto**: o CI fornece a capacidade de tomar decisões eficazes, ajudando a fornecer coragem para inovar em novas melhorias. A tomada decisão é embasada no fornecimento de informações *just-in-time* sobre o status de construção recente e as métricas de qualidade, tornando possível perceber tendências de sucesso ou falha de construção, qualidade geral e outras informações pertinentes ao projeto;

- **Estabelecer maior confiança no produto de software da equipe de desenvolvimento**: a aplicação eficaz das práticas de CI pode fornecer maior confiança na produção de um produto de software, pois o sistema de CI informa quando algo dá errado, por isso, os desenvolvedores e outros membros da equipe têm mais confiança para fazer alterações.

2.4 ENTREGA CONTÍNUA

A Entrega Contínua, em inglês *Continuos Delivery* (CD) é uma disciplina de desenvolvimento de software na qual o código que foi integrado esteja apto para ser disponibilizado no ambiente de produção a qualquer momento (LAUKKANEN; ITKONEN; LASSENIUS, 2017). No entanto, o *deploy* em produção não é automático, sendo uma decisão do líder do projeto essa decisão.

O CD consiste em vários estágios que verificam se o software está em condições de liberação (LAUKKANEN; ITKONEN; LASSENIUS, 2017), visando garantir que um aplicativo esteja sempre no estado pronto para produção após passar com sucesso em testes automatizados e verificações de qualidade. O CD emprega um conjunto de práticas, além das ações e testes

previstos na integração contínua, processos adicionais e necessários para que a modificação seja disponibilizada automaticamente para um ambiente semelhante à produção (SHAHIN; ALI BABAR; ZHU, 2017).

O objetivo da entrega contínua é poder colocar alterações de todos os tipos — incluindo novos recursos, alterações de configuração, correções de bugs e experimentos — em produção ou nas mãos dos usuários, com segurança e rapidez de maneira sustentável (HUMBLE, 2018).

Humble e Farley (2014), afirmam que a entrega contínua fornece às empresas a capacidade de entregar valor de forma rápida, confiável e repetida a clientes de baixo risco com sobrecarga manual mínima.

Para Rodríguez et al. (2017), o conceito central na abordagem de CD é um pipeline de implantação que estabelece um processo automatizado de ponta a ponta para garantir que o sistema funcione em nível técnico, execute testes de aceitação bastante automatizados e, por fim, seja implantado em um ambiente de produção ou de teste.

A Implantação Contínua, em inglês *Continuos Deployment* (CDE) é um termo usado na maior parte da literatura de forma intercambiável com o CD (RODRÍGUEZ et al., 2017), no entanto, ela é considerada uma extensão do CD (LAUKKANEN; ITKONEN; LASSENIUS, 2017).

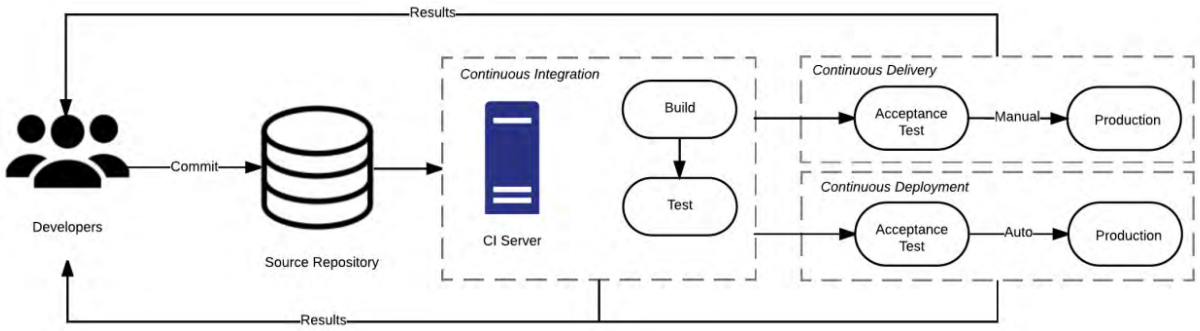
O CDE é a prática em que cada alteração é construída, testada e implantada para produção automaticamente (LAUKKANEN; ITKONEN; LASSENIUS, 2017), enquanto o CD é uma capacidade organizacional que garante que cada mudança possa ser implantada na produção, se for direcionada (no entanto, a organização pode optar por não fazê-lo, geralmente por motivos comerciais) (RODRÍGUEZ et al., 2017).

A principal diferença entre a entrega e a implantação está relacionada ao momento da disponibilização do software em produção. Ao contrário do CD, não há etapas ou decisões manuais entre um *commit* do desenvolvedor e uma implantação de produção, o processo é totalmente automatizado (LAUKKANEN; ITKONEN; LASSENIUS, 2017), como pode ser verificado na figura Figura 3.

Para Humble (2018) a prática de liberar continuamente todas as boas compilações de seu software pode ser aplicada em qualquer domínio. Portanto, Humble e Farley (2014) elencam três condições para se ter uma entrega contínua:

- Executar todo o conjunto de testes da construção da história validando-a verificando se a mesma está entregando o valor comercial esperado e que nenhuma regressão foi introduzida, o que significa ter testes automatizados abrangentes na unidade, componente e

Figura 3 – Relação entre Integração, Entrega e Implantação Contínua



Fonte: Shahin, Ali Babar e Zhu (2017)

nível de aceitação;

- A história foi apresentada aos clientes em um ambiente similar ao de produção;
- Não há problemas para disponibilizar a versão aos usuários, que significa que foram testadas características multifuncionais como capacidade, disponibilidade e segurança.

2.4.1 Práticas e Princípios da Entrega Contínua

Humble e Farley (2014), descrevem alguns princípios e práticas:

- **Criar um processo de confiabilidade e repetitividade de entrega de versão:** A repetitividade e confiabilidade derivam dessas duas princípios: automatizar quase tudo e manter tudo sob controle de versão;
- **Automatize quase tudo:** O objetivo é que os computadores executem tarefas simples e repetitivas, como testes de regressão, para que os humanos possam se concentrar na solução de problemas. Portanto, devem ser automatizados processos de implantação, testes de aceitação, *upgrades* e *downgrades* do banco de dados, entres outros processos;
- **Mantenha tudo sob controle de versão:** Tudo que é necessário para compilar, configurar, implantar, testar e entregar uma versão da aplicação deve ser mantido em algum tipo de sistema de versionamento;
- **Se é difícil, faça com mais frequência e amenize o sofrimento:** Esse princípio refere-se a quanto mais se repete um processo mais confiança se ganha. Por exemplo,

se implantar é algo difícil e custoso, deve-se ter como objetivo realizar a implantação sempre que uma mudança passar pelos testes automatizados;

- **A qualidade deve estar presente desde o início:** Construa com qualidade desde o início, pois quanto mais cedo conseguir identificar defeitos, mais barato é para corrigi-los. E é ainda mais barato corrigi-los se eles sequer entrarem no controle de versão;
- **Pronto quer dizer versão entregue:** uma funcionalidade está "pronta" quando ela foi demonstrada com sucesso e experimentada por por representantes da comunidade de usuários, em um ambiente similar à produção.
- **Todos são responsáveis pelo processo de entrega:** todos trabalham juntos para atingir as metas de nível organizacional, em vez de otimizar o que é melhor para sua equipe ou departamento, ou seja, uma equipe falha ou tem sucesso como um time e não como indivíduos. Todos podem ver rapidamente o estado da aplicação, suas condições, as várias compilações, que testes passaram e o estado dos ambientes que foram implantados;
- **Melhoria contínua:** A cada entrega o time deve se reunir e refletir sobre o que funcionou bem, o que não funcionou, e discutir ideias sobre como melhorar as coisas. As aplicações devem evoluir assim como o processo de entrega.

2.4.2 Benefícios da Entrega Contínua

O principal benefício da entrega contínua é que ela cria um processo de entrega confiável, previsível e passível de repetição, que, por sua vez, gera grandes reduções no tempo de ciclo e na entrega de novas funcionalidades, bem como a redução do risco de implantação e correções aos usuários rapidamente (CHEN, 2017; HUMBLE; FARLEY, 2014; SHAHIN; ALI BABAR; ZHU, 2017; RODRÍGUEZ et al., 2017).

Rodríguez et al. (2017) elencam que os benefícios experimentados ao aplicar o CD incluem menor tempo de colocação no mercado (*time-to-market*), *feedback* contínuo e instantâneo, especialmente dos clientes ao usar sistemas adequados de monitoramento e experimentação, confiabilidade de lançamento aprimorada, parcialmente como resultado de um foco de teste mais restrito, maior satisfação do cliente e produtividade do desenvolvedor, inovação rápida e foco de teste mais estreito.

Para Fowler (2013), os principais benefícios da entrega contínua são:

1. **Risco de implantação reduzido:** como são implantadas alterações menores, há menos erros, o que torna mais fácil a correção caso ocorra um problema.(SHAHIN; ALI BABAR; ZHU, 2017)

2. **Progresso crível:** a equipe consegue acompanhar o progresso acompanhando o trabalho feito. Uma tarefa só é considerada "pronta", quando ela foi amplamente testada e implantada em um ambiente de produção (ou semelhante à produção).

3. ***Feedback* do usuário:** o maior risco para qualquer esforço de software é que o desenvolvedor acabe construindo algo que não é útil. Quanto mais cedo e com mais frequência a equipe colocar o software em funcionamento na frente de usuários reais, mais rápido você obterá *feedback* para descobrir o quão valioso ele realmente é.

2.5 SÍNTESE DO CAPÍTULO

Este capítulo apresentou os conceitos e fundamentos teóricos usados como base para esta pesquisa.

Primeiramente, foi explicitado a diferença entre implantação e *release* de software, exemplificado o processo de entrega tradicional e a transição e conceituação do pipeline de desenvolvimento de software.

Em seguida, foi apresentada a definição do termo DevOps, sendo contada uma breve história de seu surgimentos, seus pilares, o *Culture, Automation, Lean Measurement* e *Sharing* (CALMS), princípios, práticas e, finalmente, foi descrito o seu ciclo de vida.

Por fim, foram apresentados os conceitos, práticas e benefícios das práticas contínuas. Vale destacar a diferenciação da entrega e implantação contínua. Embora sejam utilizadas como sinônimos, são práticas distintas.

A implantação contínua implica que qualquer modificação, após todos os testes automatizados passarem será implantada em produção, enquanto na entrega contínua essa implantação pode ou não acontecer, ou seja, não é um processo automático.

No capítulo a seguir, serão apresentados os trabalhos relacionados a esta pesquisa.

3 TRABALHOS CORRELATOS

A Integração e Entrega Contínua pode ser utilizada em diversas aplicações como sistemas web, *desktop*, embarcados, aplicações *mobile*, *firmware*, entre outros, apresentado diferenças em cada um dos casos. Neste capítulo serão apresentados os trabalhos correlatos a esta pesquisa. O capítulo está estruturado conforme as seguintes seções:

1. **Descrição dos Trabalhos** (seção 3.1): esta seção apresenta trabalhos relacionada a aplicação, relatos de experiência e/ou que analisem os fatores de sucesso envolvidos na implantação de integração e entrega contínua, descrevendo o objeto de estudo e as principais contribuições.

2. **Análise Comparativa** (seção 3.2): apresenta uma análise comparativa entre os trabalhos correlatos a esta pesquisa, apresentando as lacunas existentes e identificando que esta pesquisa se difere na questão do objeto de estudo.

Durante as etapas deste trabalho, buscou-se por estudos que tivessem perspectivas comuns com o tema estudado e que pudessem contribuir no seu desenvolvimento, tomada de decisões e nas escolhas das etapas que auxiliassem a atingir os objetivos propostos. O primeiro critério de seleção foi o período das pesquisas, limitando-se a trabalhos desenvolvidos nos último 10 anos, por entender que se tratar do período em pesquisadores e profissionais estavam prestando mais atenção às práticas contínuas (SHAHIN; ALI BABAR; ZHU, 2017). Os outros critérios foram os objetos de estudo, procurando os trabalhos que estudassem a adoção da integração e entrega contínua.

As principais fontes de pesquisa foram os estudos primários extraídos da revisão sistemática, as referências bibliográficas citadas em outros trabalhos e o Google Scholar[1]. Após análise das seções resumo, introdução, objetivos e conclusão, foram selecionados 14 trabalhos correlatos, apresentados na próxima seção.

3.1 DESCRIÇÃO DOS TRABALHOS

A seguir são apresentados em ordem cronológica os trabalhos correlatos escolhidos, descrevendo objeto de estudo as principais contribuições.

[1] https://scholar.google.com.br

Como estudos relevantes na área, podemos citar o de Gomede, Silva e Barros (2015), onde os autores apresentam um relato de experiência da implantação de um processo de entrega contínua em uma organização da indústria financeira. No estudo são descritas as etapas do processo de entrega e implantação do software, os ambientes utilizados e as ferramentas. No pipeline apresentado possui a etapa *Quality Assurance* (QA), com equipe especializada, de *staging* e por fim a produção. Por fim, os autores realizaram a classificação do nível de maturidade do processo de entrega de software e apresentaram 4 lições aprendidas: (1) treinas a equipe antes; (2) projetar o processo antes de utilizar uma abordagem tentativa e erro; (3) deixar claro a todos os envolvidos os benefícios; e (4) promover mudança de cultura.

Já Rossi et al. (2016), apresentam um estudo de caso sobre a implantação contínua (CDE) aplicada a softwares de dispositivos móveis no Facebook. Com a adoção do CDE houve uma redução do ciclo de *release* de 8 semanas para 1 semana para o Android.

Para os autores, a liberação de uma versão de software para plataformas móveis é mais difícil, pois não é possível reverter/avançar em caso de problema, como acontece com o software em nuvem. Para resolver esse problema, foram utilizadas *feature flags* para habilitar ou desabilitar uma funcionalidade remotamente, para que a funcionalidade com *bugs* não exija uma nova versão.

Na aplicação do CDE, foram usadas versões *alpha* e *beta* para reduzir o risco de lançar funcionalidades com *bugs*. Os autores destacam padrões que levam a softwares de qualidade inferior, como por exemplo, ter muitos desenvolvedores trabalhando no mesmo arquivo simultaneamente, códigos com baixa qualidade.

Chen (2017) apresenta um relato de experiência industrial em uma empresa multibilionária chamada Paddy Power. A empresa faz parte da indústria de apostas, oferecendo seus serviços em mercados regulamentados, por meio de casas de apostas, telefones e internet.

O relato foi realizado após 4 anos de processo de implementação do CD. Como resultado da implementação do CD, o autor elenca os benefícios alcançados como a aceleração do tempo de lançamento no mercado, construção do produto certo, melhor produtividade e eficiência, lançamentos confiáveis, melhor qualidade do produto e melhor satisfação do cliente.

Durante os 4 anos de implantação o autor encontrou diversos desafios, e para superar os desafios da adoção foram construídas 6 estratégias: (1) vender CD como analgésico; (2) estabelecer uma equipe dedicada com membros multidisciplinares; (3) entrega contínua de entrega contínua; (4) começando com as aplicações fáceis, mas importantes; (5) esqueleto de pipeline de CD visual; e (6) queda de especialista.

Elberzhager et al. (2017) apresentam um relato de experiência da implantação de práticas DevOps para um produto da Fujitsu EST, a qual foi acompanhada por aproximadamente um ano. Eles apresentam quatro perguntas que uma organização precisa responder antes de iniciar as atividades do DevOps em uma escala maior.

As lições aprendidas do estudo são: (i) o esclarecimento da direção da jornada do DevOps no início é uma etapa indispensável para garantir um procedimento estruturado e direcionado a objetivo; (ii) um entendimento comum do DevOps facilita a determinação da direção da jornada do DevOps; (iii) Começar com pequenas mudanças e selecionar um contexto gerenciável suporta a introdução do DevOps; (iv) é necessário reunir requisitos claros para as ferramentas e procedimentos a serem usados no ambiente do DevOps; (v) é necessário definir critérios para medir o sucesso; (vi) não apenas Dev e Ops, mas o gerenciamento e outras partes interessadas devem estar envolvidos; e (vii) várias práticas de DevOps servem para obter alta qualidade.

Por fim, os autores ressaltam que o nível de maturidade de uma empresa em práticas e ambientes de desenvolvimento ágil é um fator decisivo na introdução do DevOps e que devem ser analisados antes de iniciar a jornada para não perder tempo e dinheiro.

Laukkanen, Itkonen e Lassenius (2017) conduziram uma revisão sistemática da literatura para identificar os problemas, causas e possíveis soluções enfrentados na adoção da entrega contínua. Eles identificaram 40 problemas, 28 causas e 29 soluções relacionadas à adoção da entrega contínua extraídos de 30 estudos publicados até fevereiro de 2015.

Os problemas foram categorizados em 7 temas: design de construção, design de sistema, integração, teste, *release*, humano e organizacional e recursos. As soluções foram sintetizadas nos mesmos temas que os problemas, com exceção do tema design de construção, pois os autores não encontraram soluções para os problemas dessa categoria.

A temática de problema com mais evidencias foram a de testes e a de integração. Os autores acreditam que essas temáticas tenham tido mais evidências, pois se relacionam diretamente com a prática do CI e, portanto, têm sido estudados há mais tempo do que outros problemas.

As causas para os problemas de adoção eram tanto internas quanto externas aos temas. Para os autores, os problemas de design do sistema podem ser vistos como causas-raiz de problemas ao adotar o CD. Além disso, eles afirmam que os problemas humanos e organizacionais parecem ser apenas sintomas de outros problemas com base nas evidências. Por fim, afirmam que deve-se focar primeiro nos problemas de design, depois nos problemas de teste e, finalmente, nos problemas de integração.

Rodríguez et al. (2017) conduziram um mapeamento sistemático para identificar, classificar

e analisar a literatura relacionada à implantação contínua no domínio de software, o que incluiu 50 estudos primários publicados entre 2001 e 2014.

As analises realizada pelos autores foram relacionadas ao contexto dos estudos, fatores que caracterizam o CDE no contexto de produtos e serviços intensivos em software, benefícios e desafios e as lacunas de pesquisa na área.

Quanto ao contexto dos estudos destaca-se o tipo dos estudos, o qual os autores identificaram que 36% eram relatórios da industria e 24% eram estudos de caso.

Em relação ao domínio da aplicação 42% dos estudos estavam relacionados a sistema web e/ou serviços baseados na internet e 24% eram relacionados a sistemas embarcados.

Já em relação as fatores que caracterizam o CDE, foram identificados 10: (1) lançamento rápido e frequente, (2) design e arquitetura de produto flexível, (3) teste contínuo e garantia de qualidade, (4) automação (de construção e teste (CI), processos de implantação/entrega/lançamento e configuração de ambientes de implantação), (5) gerenciamento de configuração, (6) envolvimento do cliente, (7) experimentação contínua e rápida, (8) atividades pós-implantação, (9) desenvolvimento ágil e enxuto de software e (10) fatores organizacionais, incluindo funções corporativas integradas, transparência e uma cultura organizacional inovadora e experimental.

Os autores encontraram diversos benefícios e desafios relacionados ao CDE, sendo um dos desafios está relacionado aos investimentos significativos no processos de implantação, bem como em mudanças na mentalidade das pessoas e na maneira geral de trabalhar das organizações. Entre os benefícios potenciais para as organizações encontrados, cita-se a redução do tempo de colocação no mercado, um aumento na satisfação do cliente com a implantação contínua de aprimoramentos valiosos de produtos e obter *feedback* imediato durante o processo de desenvolvimento.

Por fim, os autores apresentam as lacunas de pesquisa na área referentes aos 10 fatores identificados, testes contínuos e QA, planejamento contínuo, automação e funções corporativas integradas.

Shahin, Ali Babar e Zhu (2017) apresentam uma revisão sistemática da literatura que classifica abordagens, ferramentas, desafios e práticas relacionadas a adoção de práticas contínuas, que foram extraídas de 69 estudos publicados entre 2004 e junho 2016.

Foram identificadas 30 abordagens e ferramentas associadas dividas em 6 grupos: 1) redução do tempo de construção e teste em Integração Contínua, em inglês *Continuous Integration* (CI); 2) aumentar a visibilidade e a conscientização sobre os resultados de construção e teste em CI; 3) suporte a testes contínuos (semi) automatizados; 4) detectar violações, vulnera-

bilidades e falhas no CI; 5) abordar questões de segurança e escalabilidade no pipeline de implantação; e 6) melhorar a confiabilidade e confiabilidade do processo de implantação.

No entanto, apenas 12 abordagens e ferramentas foram integradas e avaliadas no pipeline de implantação. As abordagens, ferramentas, desafios e práticas identificadas foram classificadas de forma que o leitor entenda quais são os desafios para a adoção de cada prática contínua, quais abordagens e práticas existem para apoiar e facilitar as práticas contínuas.

Após a classificação, eles identificaram 7 fatores que impactam o sucesso das práticas contínuas, em ordem de importância: "testes (esforço e tempo)", "conscientização e transparência da equipe", "bons princípios de design", "cliente", "equipe altamente qualificada e motivada", "domínio da aplicação" e "infraestrutura apropriada".

Cruz e Albuquerque (2018) apresenta um estudo de caso da adoção de DevOps em sistemas legados, sendo realizado em uma instituição financeira durante 2 anos, de médio a grande porte, a qual possuía equipes de desenvolvimento, infraestrutura e qualidade, que eram responsáveis pelo processo de implantação, sendo que o mesmo demandava muito esforço e havia grande burocracia.

Foram realizados estudos sobre os benefícios da aplicação práticas ágeis e Entrega contínua, e sobre as ferramentas a serem utilizadas. Foi proposto um processo estruturado de introdução ao DevOps e as modificações necessárias para se adaptar a sistemas legados, visando a melhoria do processo de desenvolvimento, com o intuito de ser usado como um passo a passo para empresas que queiram adotar as práticas DevOps e possuam sistemas legados.

Humble (2018) apresenta dois estudos de caso: Entrega Contínua, em inglês *Continuos Delivery* (CD) com *firmware* na HP e com com *Mainframes* na Suncorp, como uma forma de comprovar que CI/CD podem ser utilizados com outros domínios de aplicações além de *websites*.

Na HP, a adoção do CD, trouxe benefícios econômicos como: os custos gerais de desenvolvimento foram reduzidos em aproximadamente 40%; os programas em desenvolvimento aumentaram cerca de 140%; os custos de desenvolvimento por programa caíram 78%; os recursos que impulsionam a inovação aumentaram oito vezes.

Na Suncorp, com a adoção do CD houve redução no tempo de teste para integrar, sendo que os defeitos encontrados tinham correções realizadas em horas ou dias, não em semanas. E, redução no tempo de lançamento de novos produtos e serviços, permitindo uma resposta mais rápida às necessidades do cliente. Por fim, O autor relata como superar obstáculos ao CD relacionados a cultura e arquitetura.

Pessôa (2018) apresenta um estudo de caso para analisar os aspectos que impactaram a entrega contínua de software em 3 órgãos do governo com um amplo setor de TI. O método de coleta dos dados do estudo foi através da aplicação de questionários, analisados e classificados através da técnica *Grounded Theory*.

Depois, a autora classificou o nível de maturidade da entrega contínua de cada órgão com base na classificação definida por Virtanen et al. (2017), a qual tomou como base o CMMI. Verificou-se que as maturidade dos órgãos com relação a integração é baixa, uma vez que as práticas adotadas resumem-se a realizar compilações e testes unitários.

Em relação ao versionamento o nível é alto, já em relação a qualidade o nível foi considerado mediano, uma vez que eles monitoraram os erros e testam o software em um ambiente de homologação, antes de ir para produção. Quanto a automatização há falhas na na implantação automatizada e na configuração do ambiente em tempo real para alguns órgãos.

Por fim, a autora constatou que há limitação por parte dos desenvolvedores no entendimento do conceito de CD e na frequência da entrega, devido a aspectos culturais relacionados ao processo de entrega do órgão.

Proulx et al. (2018) conduziram uma revisão sistemática da literatura afim de identificar os desafios e as soluções relacionadas à CDE e, classificar quais soluções podem ser aplicadas a quais desafios, analisando 31 artigos publicados entre 2015 e 2018.

Foram descobertos 22 problemas os quais foram agrupados dentro das categorias: Humano e Organizacional, Processos, Ferramentas, Infraestrutura, Arquitetura de Aplicativos e Testes. As 19 soluções encontradas foram agrupadas dentro das categorias: Humano e Organizacional, Arquitetura, Processos e Ferramentas. Os autores associaram soluções a 14 dos 22 desafios encontrados, sendo que as categorias com mais desafios que não puderam ser associados a uma solução são Infraestrutura, Arquitetura de Aplicativos e Testes.

Siqueira et al. (2018) apresentam um relato de experiência sobre o uso de Entrega Contínua (CD) durante uma parceira acadêmica entre o Ministério do Planejamento, Orçamento e Gestão (MPOG) do governo brasileiro e duas universidades públicas para modernizar o Portal Software Público Brasileiro (SPB), projeto esse que teve duração de 30 meses.

Os autores explicitam a implantação do CD em uma grande instituição com valores tradicionais e desenvolvimento cascata, superando as dificuldades que apareciam, a burocracia e a falta de confiança do MPOG para com a equipe das universidades. Eles ressaltam como benefícios do CD, a criação da confiança entre o governo e a equipe parceira, chegando ao ponto de os funcionários do governo perceberam que seria benéfico para o projeto se eles

concedessem acesso à infraestrutura à equipe das universidades.

Durante os 30 meses, foram implantadas 84 versões, sendo que 64% dessas versões foram implantadas após a criação da equipe de DevOps. Os autores sugerem o uso de um pipeline de CD como um meio de ganhar confiança em projetos de desenvolvimento de software com o governo e grandes organizações.

Sousa (2019) apresenta um estudo de caso em uma empresa europeia da área de telecomunicações para analisar o processo de adoção do DevOps evidenciando ferramentas e técnicas utilizadas além dos principais benefícios e barreiras à sua utilização.

No estudo de caso, foi possível verificar o antes e o depois da adoção do DevOps. O autor comenta que não existe um guia, nem um caminho certo para adotar DevOps, pois a adoção deve ser efetuada através de um estudo dos processos que devem sofrer alterações, bem como das ferramentas a serem adotadas.

Foram listadas as ferramentas implementadas, bem como os benefícios da adoção o quais obteve-se o aumento da comunicação entres as equipes, bem como a frequência e melhoria dos tempos de entrega e resolução de problemas. Como barreira, o autor destaca a resistência à mudanças, falta de comunicação, falta de conhecimento técnico das equipes envolvidas relacionadas as ferramentas implementadas.

Lopes (2020) apresenta um relato de experiência da utilização da integração contínua aplicada ao Sistema de Suporte para Decisão e Transferência em Agrotecnologia (DSSAT).

O DSSAT realiza Xulações de crescimento, desenvolvimento e rendimento em função da dinâmica solo-planta-atmosfera de mais de 42 culturas como soja, milho, trigo, sorgo, entre outras. Ele é utilizado por mais de 14 mil tipos de pessoas em mais de 150 países. O código do DSSAT está disponível no Github para pessoas autorizadas.

A pesquisa relata que foram testadas Jenkins, TravisCI e o CircleCI como ferramentas de CI, o RCroptest para rodar os testes automáticos empacotados em um container Docker. Por fim a autora recomenda o TravisCI como ferramenta de CI, por ser relativamente rápido de começar a ser utilizado por sua integração com o GitHub.

O Quadro 1 mostra um resumo dos principais trabalhos relacionados com esta pesquisa:

Quadro 1 – Resumo dos principais trabalhos relacionados

Estudo	Objetivo	Escopo do Estudo
Gomede, Silva e Barros (2015)	Relato de experiência da implantação de um processo de entrega contínua em uma organização da indústria financeira	Industria
Rossi et al. (2016)	Estudo de caso sobre a implantação contínua (CDE) é aplicada a softwares de dispositivos móveis no Facebook	Indústria
Chen (2017)	Relato de experiência industrial em uma empresa multibilionária chamada Paddy Power	Indústria
Elberzhager et al. (2017)	Relato de experiência da implantação de práticas DevOps para um produto da Fujitsu EST	Indústria
Laukkanen, Itkonen e Lassenius (2017)	Revisão sistemática da literatura para identificar os problemas, causas e possíveis soluções enfrentados na adoção da entrega contínua	Academia
Rodríguez et al. (2017)	Mapeamento sistemático para identificar, classificar e analisar a literatura relacionada à implantação contínua no domínio de software	Academia
Shahin, Ali Babar e Zhu (2017)	Revisão sistemática da literatura que classifica abordagens, ferramentas, desafios e práticas relacionadas a adoção de práticas contínuas	Academia
Cruz e Albuquerque (2018)	Estudo de caso de adoção de DevOps em sistemas legados de uma instituição financeira	Indústria
Humble (2018)	Estudo de caso: CD com *firmware* na HP e com com *Mainframes* na Suncorp, como uma forma de comprovar que CI/CD podem ser utilizados com outros domínios de aplicações além de *websites*	Indústria
Pessôa (2018)	Estudo de caso para analisar os aspectos que impactaram a entrega contínua de software em 3 órgãos do governo	Indústria e Academia

Proulx et al. (2018)	Revisão sistemática da literatura afim de identificar os desafios e as soluções relacionadas à CDE e, classificar quais soluções podem ser aplicadas a quais desafios	Academia
Siqueira et al. (2018)	Relato de experiência do uso de entrega contínua durante uma parceira acadêmica entre o MPOG do governo brasileiro e duas universidades públicas	Indústria e Academia
Sousa (2019)	Estudo de caso em uma empresa europeia da área de telecomunicações para analisar o processo de adoção do DevOps	Indústria
Lopes (2020)	Relato de experiência da utilização da integração contínua aplicada ao DSSAT	Academia
Este estudo	Propor abordagens para adoção e gerenciamento de risco do pipeline CI/CD, construídas a partir de uma revisão sistemática da literatura que tinha por objetivo identificar os desafios, melhores práticas e ferramentas relacionadas a adoção da integração e entrega contínua. Também é apresentado um relato de experiência da adoção do pipeline CI/CD no IFAC	Indústria e Academia

Fonte: Elaborado pelo autor (2022)

3.2 ANÁLISE COMPARATIVA

Esta seção apresenta uma análise comparativa entre os trabalhos correlatos e esta pesquisa. Primeiramente, se faz necessário dividir os trabalhos relacionados em dois grupos, os que apresentam revisões e mapeamentos sistemáticos e os que apresentam relatos de experiência e estudos de caso. Esta pesquisa difere dos trabalhos correlatos, principalmente na questão do objeto de estudo.

Com relação aos trabalhos que apresentam revisão ou mapeamento sistemático, embora estudem desafios, melhores práticas e ferramentas relacionadas a adoção de práticas contínuas, com a exceção de Rodríguez et al. (2017), todos fizeram o relacionamento somente as práticas

com os desafios. Por este motivo, este trabalho usou como base as do Laukkanen, Itkonen e Lassenius (2017), do Shahin, Ali Babar e Zhu (2017) e Proulx et al. (2018), para realizar a atualização das evidências.

A revisão realizada teve como objetivo identificar desafios, melhores práticas e ferramentas relacionadas a adoção de práticas contínuas.

O Quadro 2, apresenta o comparativo deste trabalho na perspectiva da revisão sistemática.

Como fruto da revisão realizada, foi construído um corpo de conhecimento que funciona como um *framework* para compor o *pipeline* concreto e efetivo a ser implementado, além de um processo de adoção de pipeline CI/CD e uma abordagens para o gerenciamento de risco do pipeline como forma de auxiliar órgãos públicos a adotarem a Integração e a Entrega Contínua como uma forma de reduzir o *time to market* e aumentar a qualidade dos produtos entregues, gerando uma redução no custo das operações, devido ao tempo de inatividade reduzido e maior velocidade de desenvolvimento.

Quadro 2 – Comparativo com as Revisões Sistemáticas

	Laukkanen, Itkonen e Lassenius (2017)	**Rodríguez et al. (2017)**	**Shahin, Ali Babar e Zhu (2017)**	**Proulx et al. (2018)**	**Este estudo**
Foco	CDE	CDE	CI, CD e CDE	CDE	CI/CD
Desafios	X	X	X	X	X
Melhores Práticas	X		X	X	X
Ferramentas			X		X
Relaciona Desafios e Melhores Práticas	X		X	X	X
Relaciona Desafios e Ferramentas					X
Relaciona Melhores Práticas e Ferramentas					X
Proposta ou abordagem para adotar práticas contínuas					X

Fonte: Elaborado pelo autor (2022)

Os trabalhos que apresentam relatos de experiência e estudos de caso, somente o Siqueira et al. (2018) apresenta um estudo voltado para instituições públicas, no entanto, tratava-se da adoção de CD em um desenvolvimento terceirizado realizado por duas universidades e não pelo próprio MPOG. Além disso, o artigo informa que o CD só foi utilizado pela equipe da

universidade que desenvolvia o software e não para todos os projetos de software da universidade.

Outro que se assemelha bastante com o objetivo do relato de experiência dessa pesquisa é o Gomede, Silva e Barros (2015), porém não é explicitado o processo de adoção e nem é apresentado os desafios que enfrentaram e foi realizado em uma empresa privada, como os demais estudos.

Diferente dos outros estudos, esta pesquisa tem como proposta de seu relato de experiência servir como estratégias minímas para a adoção do pipeline CI/CD em uma instituição de ensino pública, apresentando as etapas percorridas, ferramentas utilizadas, desafios superados e lições aprendidas. A instituição possui a equipe de desenvolvimento pequena e não possui uma equipe responsável pela qualidade/testes do produto.

O Quadro 3 apresenta o comparativo deste trabalho na perspectiva do relato de experiência

Como lacunas de pesquisa, identificamos que embora todos os trabalhos correlatos tenham uma natureza prática, nenhum deles apresentou abordagem e/ou processos para apoiar a adoção e/ou gerenciar os riscos do pipeline CI/CD, e nem relatos ou estratégias que apresentassem etapas percorridas, ferramentas utilizadas, desafios superados e lições aprendidas que pudessem servir de guia para outra instituição adotar as práticas de integração e entrega contínua.

Portanto, este trabalho procura suprir estas lacunas, apresentando um corpo de conhecimento que funciona como um *framework* para compor o pipeline a ser implementado, um processo de adoção, uma abordagem para o gerenciamento de risco e estratégias minímas para implantar um pipeline CI/CD.

Quadro 3 – Comparativo com os Relatos de Experiência

Proposta	Foco	Etapas Percorridas	Descrição do Pipeline	Ferramentas	Benefícios	Desafios	Lições Aprendidas
Gomede, Silva e Barros (2015)	CDE		X	X	X		X
Rossi et al. (2016)	CDE		X		X		
Chen (2017)	CD	X			X	X	X
Elberzhager et al. (2017)	DevOps		X				X
Cruz e Albuquerque (2018)	DevOps	X	X	X		X	
Humble (2018)	CD				X	X	
Pessôa (2018)	CD		X	X			
Siqueira et al. (2018)	CD		X	X	X	X	X
Sousa (2019)	DevOps			X	X	X	
Lopes (2020)	CI		X	X	X		
Este estudo	CI/CD	X	X	X	X	X	X

Fonte: Elaborado pelo autor (2022)

3.3 SÍNTESE DO CAPÍTULO

Neste capítulo foram apresentados 14 trabalhos correlatos a esta pesquisa que tinham como foco a análise dos fatores de sucesso envolvidos na implantação de integração e entrega contínua, descrevendo o objeto de estudo e as principais contribuições. Ao final, foi apresentada uma análise comparativa desta pesquisa com os trabalhos relacionados, apresentando as lacunas existentes e identificando que esta pesquisa se difere na questão do objeto de estudo.

Para realizar a análise comparativa, os trabalhos correlatos foram divididos em dois gru-

pos, os que apresentam revisões e mapeamentos sistemáticos e os que apresentam relatos de experiência e estudos de caso. A diferença desta pesquisa com relação aos estudos do grupo das revisões está na associação das melhores práticas e ferramentas aos desafios e das melhores práticas e ferramentas, o que gerou um corpo de conhecimento que funciona como um *framework* para compor o *pipeline* concreto e efetivo a ser implementado. O que tornou possível a criação de um processo de adoção de pipeline CI/CD e de abordagem para gerenciamento de risco do pipeline. Já a diferença com os estudos do grupo dos relatos de experiência, a diferença está no objetivo do relato, que nesta pesquisa é apresentar estratégias minímas para a adoção do pipeline CI/CD em uma instituição de ensino pública, contendo as etapas percorridas, ferramentas utilizadas, desafios superados e lições aprendidas.

No capítulo a seguir, serão apresentados os procedimentos metodológicos utilizados nesta pesquisa, com detalhamento de sua abordagem e instrumentos aplicados.

4 METODOLOGIA DE PESQUISA

Metodologia científica é necessária, entre outras razões, para tornar os resultados da pesquisa mais confiáveis e possíveis de serem reproduzidos, de forma independente, por outros pesquisadores. Este capítulo apresenta a abordagem metodológica selecionada para esta pesquisa. O capítulo está estruturado conforme as seguintes seções.

1. **Classificação da Pesquisa** (seção 4.1): esta seção apresenta a classificação da pesquisa com o quadro metodológico definido.
2. **Ciclo da Pesquisa** (seção 4.2): as etapas da pesquisa são detalhadas nesta seção, assim como o processo da revisão sistemática com todos os passos que se seguiram para a realização da mesma. E por fim, o procedimento usado para a análise e organização dos resultados.

4.1 CLASSIFICAÇÃO DA PESQUISA

Esta pesquisa adota a abordagem **indutiva** baseada em dados de natureza **qualitativa**, os quais foram coletados através da **revisão sistemática da literatura**. O Quadro 4 apresenta o quadro metodológico resumido da pesquisa.

Quadro 4 – Classificação da Pesquisa

Quadro Metodológico	
Método de Abordagem	Indutivo
Método de Procedimento	Revisão Sistemática da Literatura
Natureza dos Dados	Qualitativa
Variáveis	**Independentes:** Boas Práticas e Ferramentas
	Dependentes: Desafios

Fonte: Elaborado pelo autor (2022)

O método de abordagem indutivo, segundo Marconi e Lakatos (2010) é um processo mental por intermédio do qual, partindo de dados particulares que sejam suficientemente constatados, infere-se uma verdade geral ou universal, não contida nas partes examinadas. Marconi e Lakatos (2010) afirmam ainda que a indução é realizada em três etapas:

- **Observação dos Fenômenos:** Nesta etapa o objetivo é descobrir as causas da sua manifestação;

- **Descoberta do relacionamento entre eles:** Busca-se descobrir a relação constante por intermédio da comparação;
- **Generalização do relacionamento:** Nesta etapa ocorre a generalização entre os fenômenos e fatos semelhantes.

O método de procedimento definido para a etapa mais concreta no processo de investigação desta pesquisa foi o de Revisão Sistemática da Literatura (RSL) (KITCHENHAM, 2007) como forma de analisar e interpretar o conjunto de dados obtidos na literatura existente referentes a uma questão de investigação particular, área temática, ou fenômeno de interesse, baseando-se em evidências. A escolha da revisão ao invés do mapeamento foi baseada na natureza definida da questão de pesquisa com o intuito de construir uma abordagem de apoio à tomada de decisão na seleção de estudos empíricos em Engenharia de Software.

A natureza das variáveis é qualitativa, portanto, apesar de que alguns resultados desta pesquisa apresentam dados quantitativos, essa pesquisa se caracteriza como uma abordagem qualitativa. Marconi e Lakatos (2010) explicam que a abordagem qualitativa é apropriada para analisar e interpretar aspectos mais profundos e detalhados do comportamento humano e ainda fornecendo análises mais detalhadas sobre as investigações, atitudes e tendências de comportamento.

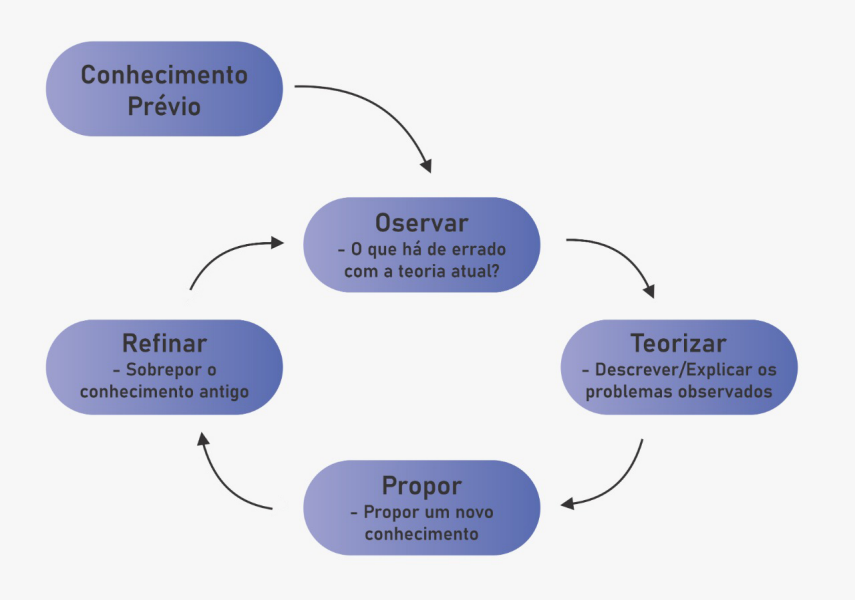
Segundo Marconi e Lakatos (2010), as variáveis de uma pesquisa podem ser consideradas independentes ou dependentes. As independentes são consideradas determinantes que fornecem condição ou causa para um resultado, influenciando, determinando ou afetando as variáveis dependentes. Com esta definição, este trabalho pretende investigar as boas práticas e ferramentas como variáveis independentes e os desafios da adoção de integração e entrega contínua como variáveis dependentes.

4.2 CICLO DA PESQUISA

As principais etapas que constituíram o ciclo dessa pesquisa são apresentadas na Figura 4.

O trabalho iniciou-se com o conhecimento prévio que se tinha da área e adquirido com base na pesquisa bibliográfica tradicional (revisão *ad hoc* com os principais temas que envolvem a pesquisa - DevOps, Práticas Contínuas e Entrega de Software) apresentada no referencial teórico, descrito no Capítulo 2 desse trabalho. As bibliografias selecionadas foram dissertações acadêmicas, artigos científicos e livros em português e em inglês.

Figura 4 – Etapas da Pesquisa



Fonte: Elaborado pelo autor, baseado em Easterbrook et al. (2006)

Posteriormente foram observados e analisados trabalhos na literatura sobre a adoção de integração e entrega contínua, e com isso verificou-se a importância do tema e a ausência de estudos que apoiem os desenvolvedores durante o processo de adoção e gestão dos desafios, apresentados no Capítulo 3.

A partir desse estudo exploratório inicial sobre a área para verificar a relevância do tema, o problema do estudo foi formulado e explicitado através das questões de pesquisa, apresentadas no Capítulo 1. Com isso, uma revisão sistemática da literatura para coleta de evidências foi planejada e executada. O primeiro passo foi a definição de um protocolo, que foi seguido durante toda a revisão. Este protocolo é apresentado resumidamente nesse capítulo e por completo no Apêndice B desse trabalho.

Ao final da revisão sistemática, é proposto um processo para adoção de um pipeline de integração e entrega contínua e uma abordagem para o gerenciamento de risco do pipeline, que reuni o conhecimento existente na área sobre o tema e propõe como novo conhecimento, uma análise que combina os desafios às propostas de soluções (melhores práticas e ferramentas). A reunião do conhecimento existente e a criação de uma abordagem que agrega uma nova forma de análise, são apresentados no Capítulo 5. Esse ciclo pode ser repetido para analisar e refinar a abordagem proposta através de novos experimentos.

De acordo com Kitchenham, Pickard e Pfleeger (1995), se o estudo é por natureza comparativo, contrastando os resultados do uso de um método com os resultados do uso de outro.

Para evitar viés e garantir a validade interna, deve-se identificar uma base válida para avaliar os resultados do estudo.

Por fim, o processo e a abordagem criados nesta pesquisa foram aplicados em um projeto piloto do Instituto Federal do Acre (IFAC) e avaliados através do *Goal Question Metric* (GQM) utilizando o método de comparação do projeto irmão. A partir dessa aplicação, foi criada uma estratégia mínima viável para adoção de integração e entrega contínua que é apresentada no Capítulo 6.

O método de comparação do projeto irmão, em inglês, *sister project*, envolve dois projetos, um que utiliza o novo método e outro que utiliza o método atual (KITCHENHAM; PICKARD; PFLEEGER, 1995). Cada projeto deve ser típico da organização e ambos devem ter características semelhantes de acordo com as variáveis de estado que foram escolhidas. Por exemplo, para cada projeto participante do estudo, mede-se a produtividade em pontos de função por hora da equipe usando o método A (o método atual) e o método B (o novo método).

4.2.1 Processo da Revisão Sistemática

A Revisão Sistemática da Literatura (RSL) (KITCHENHAM, 2007) foi conduzida como forma de atualizar as revisões do Laukkanen, Itkonen e Lassenius (2017), do Shahin, Ali Babar e Zhu (2017) e Proulx et al. (2018), no intuito de encontrar e analisar o maior número de trabalhos primários relevantes e reconhecidos na área que pudessem responder as questões de pesquisa. Essa seção discute alguns tópicos do protocolo de pesquisa que guiou a revisão, o qual encontra-se completo no Apêndice B.

4.2.1.1 Questões de pesquisa

Com o objetivo "o que muda no desenvolvimento de software quando são implantadas práticas de integração/entrega contínua?" e "como apoiar o uso dessas práticas?" a pesquisa parte para três questões de investigação mais específicas que possam responder essas perguntas na busca por uma abordagem que apoie com práticas e ferramentas eficazes a implantação de integração e entrega contínuas.

As seguintes questões de pesquisa foram utilizadas para orientar a análise dos dados relacionados ao problema abordado nesta pesquisa:

(Q1) Quais desafios foram relatados para a adoção de práticas contínuas?

(Q2) Que práticas foram relatadas para implementar com êxito práticas contínuas?

(Q3) Quais ferramentas foram empregadas para projetar e implementar pipelines de implantação?

4.2.1.2 Estrutura das Questões

Kitchenham (2007) recomenda considerar as questões de pesquisa a partir da seguinte estrutura PICOC (Population, Intervention, Context, Outcomes, e Comparison) que traduzida para o português seria: População, Intervenção, Contexto , Resultados e Comparação. Para cada pergunta da pesquisa, são apresentados, a seguir, os elementos PIO (Population, Interventaion, e Outcome):

Q1:

- **População** (P): Software ou desenvolvimento de software
- **Intervenção** (I): Adoção de práticas contínuas
- **Resultado** (O): Desafios

Q2:

- **População** (P): Software ou desenvolvimento de software
- **Intervenção** (I): Práticas
- **Resultado** (O): Implementar com êxito práticas contínuas

Q3:

- **População** (P): Software ou desenvolvimento de software
- **Intervenção** (I): Ferramentas
- **Resultado** (O): Projetar e implementar pipelines de implantação

A Comparação e o Contexto da estrutura PICOC não foram utilizados, uma vez que os objetivos do trabalho não incluem nenhum contexto específico e não buscam a comparação entre os tópicos investigados.

4.2.1.3 Estratégia de Busca

A construção dos termos de busca foi realizada seguindo os seguintes passos:

1. A partir das estruturas das questões de investigação (PIO) definidas anteriormente, os principais termos são identificados;
2. É realizada a tradução desses termos para o inglês por ser a língua utilizada nas bases de dados eletrônicas pesquisadas e nas principais conferências e jornais dos tópicos de investigação;
3. Sinônimos são identificados com a orientação de um especialista no tema de investigação para cada um dos principais termos;
4. As strings de busca são geradas a partir das estruturas das questões e da combinação dos termos chave e sinônimos. São usados OR (ou) entre os sinônimos identificados e AND (e) entre os termos chaves. Algumas adaptações são necessárias de acordo com as necessidades especificas de cada base de dados. Devido a esta revisão tratar-se de uma atualização, o resultado da execução da string foi filtrado por publicações a partir de 2016. Possíveis peculiaridades das bibliotecas digitais e adaptações mediante a isso são registradas.

Os termos e sinônimos identificados são apresentados abaixo:

Integração contínua: Continuous integration, Rapid integration, Fast integration, Quick integration, Continuous build, Rapid build, Fast build, Quick build;

Entrega contínua: Continuous delivery, Rapid delivery, Fast delivery, Quick delivery;

Implantação contínua: Continuous deployment, Rapid deployment, Fast deployment, Quick deployment, Continuous release, Rapid release, Fast release, Quick release;

Práticas contínua: Continuous practice;

Engenharia contínua: Continuous software engineering, Continuous engineering

Software: Software, Program, System, Application Product

O Quadro 5 apresenta a string de busca gerada para o trabalho, visto que as revisões base para esta usaram apenas uma string para coletar os artefatos.

Quadro 5 – String de busca da pesquisa

String para a Pesquisa
(“continuous integration” OR “rapid integration” OR “fast integration” OR “quick integration” OR “continuous delivery” OR “rapid delivery” OR “fast delivery” OR “quick delivery” OR “continuous deployment” OR “rapid deployment” OR “fast deployment” OR “quick deployment” OR “continuous release” OR “rapid release” OR “fast release” OR “quick release” OR “continuous build” OR “rapid build” OR “fast build” OR “quick build” OR “continuous practice” OR “continuous practices” OR “continuous software engineering” OR “continuous engineering”) AND (software* OR program* OR system* OR application* OR product*)

Fonte: Elaborado pelo autor (2022)

4.2.1.4 Fontes de Busca

Os critérios para a seleção das fontes foram: (1) disponibilidade de consultar os artigos na web; (2) presença de mecanismos de busca usando palavras-chave; e, (3) importância e relevância das fontes. As fontes de pesquisa utilizadas para a busca dos estudos primários são listadas abaixo:

- *IEEEXplore Digital Library* (https://ieeexplore.ieee.org/)
- *ACM Digital Library* (https://portal.acm.org)
- *Elsevier Scopus* (https://www.scopus.com/)
- *Elsevier ScienceDirect* (www.sciencedirect.com)

Outras fontes foram inicialmente consideradas como potenciais para as buscas: *Google*, *Google Scholar*, *SpringerLink*, *ISI Web of Science* e *Wiley Online Library*. Entretanto, estas foram posteriormente excluídas da lista final de fontes por algumas das seguintes razões:

- Algumas por não estarem presentes em importantes revisões sistemáticas ou não terem sido recomendadas por especialistas;
- Algumas por não permitirem a visualização ou download dos trabalhos sem pagamento ou licenças que a instituição de realização do trabalho não possuía;
- Algumas por já serem indexadas por algumas das fontes já listadas na pesquisa.

4.2.1.5 Critérios de Inclusão e Exclusão dos Estudos

A inclusão de um trabalho é determinada pela relevância em relação às questões de investigação, determinada pela análise do titulo, palavras-chave, resumo, introdução e conclusão. Os seguintes critérios de inclusão foram definidos:

a) Estudos que tratem primariamente ou secundariamente de ferramentas associadas para facilitar as práticas continuas;

b) Estudos que tratem primariamente ou secundariamente de desafios na adoção de práticas contínuas;

c) Estudos que tratem primariamente ou secundariamente Boas Práticas, Lições Aprendidas e Fatores de Sucesso relacionados à Integração ou Entrega Contínua;

A partir também da análise do titulo, palavras-chave, resumo, introdução e conclusão, são excluídos os estudos que se enquadrem em qualquer dos casos abaixo:

a) Estudos que não respondam nenhuma das questões de pesquisa;

b) Estudos que não estejam disponíveis livremente para consulta na web ou Portal da Capes;

c) Estudos que apresentem texto, conteúdo e resultados incompletos, ou seja, trabalhos com resultados não concluídos não serão aceitos;

d) Estudos claramente irrelevantes para a pesquisa, de acordo com as questões de investigação levantadas;

e) O foco principal do artigo ser avaliar uma nova tecnologia ou ferramenta em um caso da vida real;

f) Não são artigos científicos revisados por pares (por exemplo, apresentações, chamadas para trabalhos, discursos, prefácios etc.) ou capítulos de livros e livros;

g) Estudos Repetidos: se determinado estudo estiver disponível em diferentes fontes de busca, a primeira pesquisa será considerada;

h) Estudos Duplicados: caso dois trabalhos aprese ntem estudos semelhantes, apenas o mais recente e/ou o mais completo será incluído, a menos que tenham informação complementar;

i) Não estar em português ou inglês;

j) *Short papers* (menos de 6 páginas);

4.2.1.6 Processo de Seleção dos Estudos Primários

De acordo com Kitchenham (2007), as buscas iniciais retornam uma grande quantidade de estudos que não são relevantes, não respondendo às questões ou mesmo não tendo relação com o tópico em questão. Então, estudos totalmente irrelevantes são descartados no início. O Quadro 6 apresenta as etapas do processo de seleção dos estudos primários.

Quadro 6 – Processo de seleção dos estudos primários

Etapas do Processo de Seleção dos Estudos Primários	
Etapa 1	O pesquisador inicialmente realiza as buscas para identificar os potenciais estudos primários e a partir da leitura dos títulos dos trabalhos que a pesquisa retorna e palavra-chave, excluem trabalhos que claramente são irrelevantes para as questões investigadas.
Etapa 2	A partir da lista com os potenciais candidatos a estudos primários, todos os trabalhos são avaliados, mediante a leitura do resumo e conclusão, considerando-se os critérios de inclusão e exclusão, para então se chegar a uma lista final de estudos primários.
Etapa 3	Os estudos incluídos são documentados através de formulários, assim como todos os trabalhos excluídos e o critério que definiu sua exclusão. Posteriormente, cada estudo primário é lido e através de formulários a extração dos dados e avaliação da qualidade dos trabalhos é realizada.

Fonte: Elaborado pelo autor (2022)

4.2.1.7 Avaliação da Qualidade

Como um complemento para os critérios gerais de inclusão e exclusão, é considerado importante avaliar a qualidade dos estudos primários (KITCHENHAM, 2004). Apesar de não existir uma definição universal do que seja qualidade de estudo, a maioria dos checklists incluem questões que objetivam avaliar a extensão em que o viés é minimizado e a validação interna e externa são maximizadas (KITCHENHAM, 2007; TRAVASSOS; BIOLCHINI, 2007).

Para a realização da avaliação da qualidade dos estudos primários, algumas questões foram definidas, as mesmas estão disponíveis no formulário de extração dos dados e podem ser visualizadas no Quadro 7 a seguir. Dentre os critérios de avaliação, existem alguns que deverão

ser aplicados a todos os tipos de estudo e outros que são específicos para cada tipo de estudo (Experimental, Teórico, Revisões Sistemáticas e Relatos de Experiência Industrial).

Os estudos experimentais são aqueles baseados em evidências diretas ou experimentos. Já os teóricos são estudos conceituais e baseados em um entendimento de uma área, referenciando outros trabalhos relacionados. Por sua vez, revisões sistemáticas da literatura avaliam estudos primários através de um processo rigoroso. E os relatos de experiência industrial apresentam um estudo baseado na experiência prática na indústria. Easterbrook et al. (2008) classifica os métodos para estudos experimentais como: Experimentos Controlados, Estudos de Caso, Pesquisa de Campo, Etnografia e Pesquisa-Ação.

Além de perguntas relacionadas à como o estudo foi conduzido e os resultados de cada trabalho avaliado, foram adicionadas 3 (três) perguntas relacionadas às questões de investigação, no intuito de verificar o quanto cada estudo atende aos objetivos desta pesquisa. Como as questões são semelhantes e complementares, um mesmo trabalho pode apresentar resultados para as 4 (quatros) questões de pesquisa e assim obter bons conceitos nos três critérios.

Para a avaliação da qualidade dos estudos é usada a escala Likert-5, que permite respostas gradativas. Para responder as questões dos critérios de qualidade. O pesquisador pode usar os seguintes níveis de concordância ou discordância (concordo totalmente, concordo parcialmente, neutro, discordo parcialmente e discordo totalmente). Para a avaliação, devem ser consideradas as seguintes observações:

- **Concordo totalmente (4):** deve ser concedido no caso em que o trabalho apresente no texto os critérios que atendam totalmente a questão;
- **Concordo parcialmente (3):** deve ser concedido no caso em que o trabalho atenda parcialmente aos critérios da questão;
- **Neutro (2):** deve ser concedido no caso em que o trabalho não deixe claro se atende ou não a questão;
- **Discordo parcialmente (1):** deve ser concedido no caso em que os critérios contidos na questão não são atendidos pelo trabalho avaliado;
- **Discordo totalmente (0):** deve ser concedido no caso em que não existe nada no trabalho que atenda aos critérios da questão.

Quadro 7 – Formulário de Avaliação de Qualidade

Item	Critérios de Qualidade	Valores
1	Os objetivos ou questões do estudo são claramente definidos (incluindo justificativas para a realização do estudo)?	
2	O tipo de estudo está definido claramente?	
Desenvolvimento		
3	Existe uma clara descrição do contexto no qual a pesquisa foi realizada?	
4	O trabalho é bem/adequadamente referenciado (apresenta trabalhos relacionados/semelhantes e baseia-se em modelos e teorias da literatura)?	
Conclusão		
5	O estudo relata de forma clara e não ambígua os resultados?	
6	Os objetivos ou questões do estudo são alcançados?	
Critério Específico para estudos Experimentais		
7	Existe um método ou um conjunto de métodos descrito para a realização do estudo?	
Critério Específico para estudos Teóricos		
7	Existe um processo não tendencioso na escolha dos estudos?	
Critério Específico para Revisões Sistemáticas		
7	Existe um protocolo rigoroso, descrito e seguido?	
Critério Específico para Relato de Experiência Industrial		
7	Existe uma descrição sobre a(s) organização(ões), equipe(s), projeto(s) e distribuição envolvida?	
Critérios para as Questões de Investigação (Q1, Q2 e Q3)		
8	O estudo lista primária ou secundariamente de desafios na adoção de práticas contínuas?	
9	O estudo lista primária ou secundariamente Boas Práticas, Lições Aprendidas e Fatores de Sucesso relacionados à Integração ou Entrega Contínua?	
10	O estudo lista primária ou secundariamente de ferramentas associadas para facilitar as práticas continuas?	
TOTAL		

Fonte: Elaborado pelo autor (2022)

Os estudos primários avaliados podem se enquadrar em 5 níveis de qualidade, corforme classificação de Beecham et al. (2008), a partir dos valores finais da avaliação de cada estudo, conforme mostra a Tabela 1.

Tabela 1 – Avaliação da Qualidade

Faixa de Notas	Avaliação
>86%	Excelente
66% - 85%	Muito Boa
46% - 65%	Boa
26% - 45%	Média
<26%	Baixa

Fonte: Elaborado pelo autor, baseado em Beecham et al. (2008)

4.2.1.8 Extração dos Dados

Para apoiar a extração e registro dos dados e posterior análise, foi utilizada a ferramenta StArt[1], que tem como objetivo auxiliar o pesquisador, dando suporte à aplicação da Revisão Sistemática. A ferramenta foi bastante útil pois possibilita a importação no formato BibTex do resultados das pesquisas nas bases de dados, o cadastro do protocolo, e dos formulários de extração de dados e de avaliação da qualidade dos estudos, permitindo que o pesquisador registre o motivo que levou a exclusão do estudo. Cada alteração feita é salva e ao final da pesquisa o arquivo serve como documentação da revisão e pode ser usado por outro pesquisador a fim de repeti-la.

Na ferramenta, para cada trabalho aprovado pelo processo de seleção, o pesquisador registrava o critério de inclusão, ao final foi extraída uma listagem com os trabalhos incluídos, com apenas as informações que identificam o trabalho e dados que serão apresentados em forma de gráficos nos resultados da revisão. Para os trabalhos exclúidos foi registrado na ferramenta o motivo que levou a exclusão. O formulário extrair as informações gerais e realização da avaliação da qualidade, foi cadastrado e preenchido na ferramenta.

4.2.1.9 Síntese dos Dados

Após a coleta dos dados, as informações devem ser tabuladas de acordo com as questões de pesquisa, as tabelas devem ser estruturadas de forma a destacar as semelhanças e diferenças entre os resultados do estudo (KITCHENHAM, 2007; TRAVASSOS; BIOLCHINI, 2007). Kitchenham (2007) afirma que a síntese dos dados pode ser quantitativa e/ou qualitativa, sendo que a primeira necessariamente seria tratada através de meta-análise. Nesta pesquisa, a natureza

[1] http://lapes.dc.ufscar.br/tools/start_tool/

dos dados é qualitativa, logo uma síntese qualitativa é realizada.

Os dados extraídos dos estudos são organizados em tabelas exportados da ferramenta StArt para uma planilha eletrônica. A partir de similaridades dos dados extraídos, utilizando-se para isso o método de comparações constantes, é realizada a síntese dos dados e são listados desafios e soluções, boas práticas e ferramentas identificadas na adoção da integração, entrega e implantação contínua para responder a cada questão de pesquisa.

O processo se inicia com a marcação de trechos dos textos dos trabalhos (ou dados qualitativos) que fornecem informação relevante para responder as questões de pesquisa. A cada um desses trechos são associados a códigos de cores que indicam que tipo, ou categoria, de informação o trecho está provendo. O procedimento de análise desses dados extraídos e sintetizados é apresentado na próxima subseção.

4.2.2 Procedimento para Análise dos Resultados

Com os dados da revisão sistemática extraídos e sintetizados, realizou-se uma análise mais detalhada dos mesmos para a criação de uma abordagem que relacione os desafios às boas práticas e identificadas.

Como todos os dados identificados como desafios, melhores práticas e ferramentas nos estudos primários eram extraídos e recebiam uma numeração. A associação dos desafios com as possíveis soluções lavaram em conta a definição e os objetivos das solução e o contexto em que foram usadas, uma vez que poucos estudos descreveram ou fizeram relação com as soluções utilizadas para superar os desafios. Os dados recebiam a seguinte formatação:

Para os Desafios:

D1 ... Dn - <Desafios>

Para os Soluções:

MP1 ... MPn - <Melhores Práticas>

F1 ... Fn - <Ferramentas>

Onde:

- D1 ... Dn - um número atribuído sequencialmente para os desafios que os estudos primários avaliados apresentavam;

- Desafio - desafios na adoção de práticas contínuas, apresentadas pelos estudos;
- MP1 ... MPn - um número atribuído sequencialmente às melhores práticas propostas pelos estudos primários;
- Melhor Prática - práticas relacionadas a adoção e uso de práticas contínuas;
- F1... Fn - um número atribuído sequencialmente às ferramentas de apoio propostas pelos estudos primários;
- Ferramenta - solução tecnologia proposta para apoiar a adoção e uso de práticas contínuas.

A síntese das evidências coletadas foi feita na forma de tabelas que relacionam a evidência com a categoria construída a partir das evidências e com a origem através da referência ao estudo primário. Nestas tabelas, também são apresentadas as frequências de ocorrência das evidências, através da contagem no número de vezes que a evidência é encontrada em estudos diferentes.

4.3 SÍNTESE DO CAPÍTULO

Esta pesquisa de abordagem indutiva baseada em dados de natureza qualitativa, desenvolveu-se através da execução de uma revisão sistemática da literatura.

Como resultado das evidencias extraídas da execução da revisão foi elaborada uma abordagem para orientar a adoção de um pipeline de implantação. Essa abordagem foi aplicada em um projeto piloto do IFAC sendo avaliada através do GQM utilizando o método de comparação do projeto irmão. As experiências dessa adoção criaram uma estratégia miníma viável para a adoção de integração e entrega contínua em um instituto federal.

Nos capítulos seguintes (4 e 5), apresentam-se os resultados da revisão sistemática, a abordagem para implantação de um pipeline de integração e entrega contínua, além da estratégia miníma viável para adota-lo, buscando oferecer corpo de conhecimento sobre o assunto para auxiliar profissionais e pesquisadores.

5 EXECUÇÃO E RESULTADOS

O Capítulo 5 apresenta os resultados da execução da revisão sistemática da literatura e sua análise, levando-se em consideração quatro partes distintas. Esses componentes constituem as principais contribuições do estudo e serão detalhados da seguinte maneira:

1. **Análise Descritiva da Revisão Sistemática** (seção 5.1): apresenta dados gerais da revisão, como: quantidade de trabalhos retornados nas buscas, processo de seleção com o número final de estudos primários, distribuição ao longo dos anos, locais de publicação, os tipos dos estudos, distribuição do domínio de aplicação, e por fim, a avaliação da qualidade;

2. **Análise das Evidências** (seção 5.2): apresenta e descreve as evidências identificadas pela revisão sistemática, isto é, apresentação dos resultados para cada questão de pesquisa (Q1, Q2 e Q3);

3. **Ameaças à Validade** (seção 5.3): apresenta as ameaças à validade da pesquisa, tratando os riscos, ameaças e possíveis vieses do estudo;

4. **Discussão sobre os Resultados Obtidos** (seção 5.4): apresenta uma análise dos principais resultados obtidos pela pesquisa.

5. **Proposta de Abordagem para Adoção de Pipeline de Entrega Contínua** (seção 5.5)– estabelece uma relação entre os resultados de cada questão de pesquisa, propondo uma abordagem para a implantação de um pipeline de entrega contínua, no qual a equipe de desenvolvimento pode identificar desafios e possíveis soluções. Para cada desafio na adoção de integração, entrega e implantação contínua (Q1), uma proposta de solução (Q2 e/ou Q3) é identificada, e para cada melhor prática (Q3), ferramentas (Q4) de apoio são relacionadas.

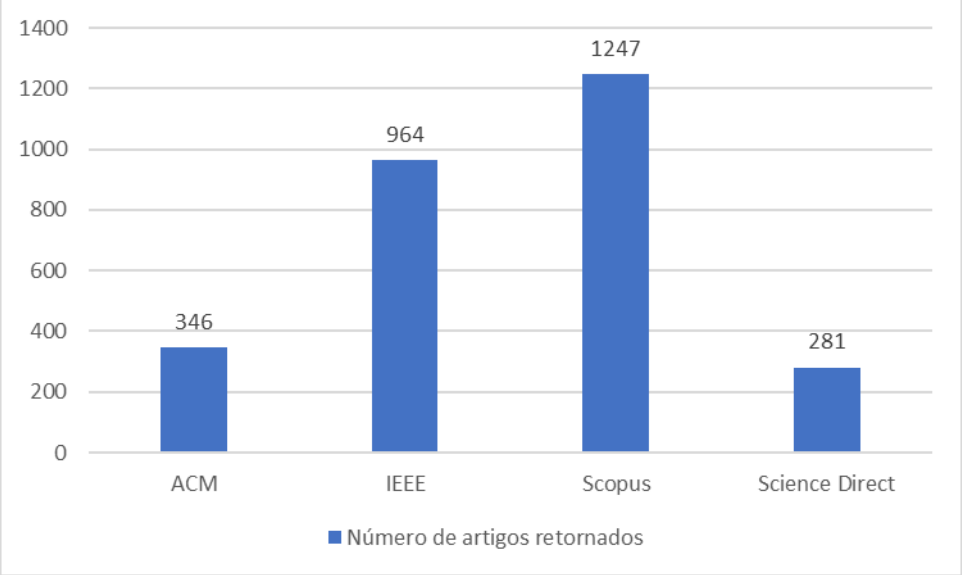
5.1 ANÁLISE DESCRITIVA DA REVISÃO SISTEMÁTICA

Argumenta-se que relatar informações demográficas sobre os tipos e locais dos artigos revisados sobre determinado tópico de pesquisa é útil para novos pesquisadores interessados em realizar pesquisas sobre esse tópico. Portanto, a informação demográfica é considerada uma das informações importantes em um RSL. (SHAHIN; ALI BABAR; ZHU, 2017)

A revisão sistemática foi executada de acordo com o que foi definido no protocolo que se encontra resumido na subseção 4.2.1 e disponível por completo no Apêndice B. Tendo em vista que esta revisão trata-se de uma atualização das revisões do Laukkanen, Itkonen e Lassenius (2017), do Shahin, Ali Babar e Zhu (2017) e do Proulx et al. (2018), foi determinado um limite do período de publicação a partir de 2016.

Após serem definidas a *strings* de busca e base de dados, as buscas primárias retornaram um total de 2.837 trabalhos, dos quais, 964 trabalhos foram identificados no IEEE, 346 na ACM, 1.247 na Scopus e 281 no ScienceDirect, conforme mostrado na Figura 5.

Figura 5 – Gráfico do número de trabalhos retornados



Fonte: Elaborado pelo autor, com dados da revisão sistemática (2022)

Como se pode verificar, o número de estudos retornados na busca primária foi alto, porém a partir do processo de seleção definido anteriormente, esse número foi bastante reduzido. O Quadro 8 apresenta a evolução, em números, do processo de seleção de estudos primários. Na busca primária para cada *string*, no total, 2.837 estudos retornados, e que a partir da primeira seleção por título, resumo e palavra-chave, foram identificados 526 estudos potencialmente relevantes para a pesquisa.

Quadro 8 – Seleção dos estudos primários

Seleção dos estudos primários				
Fontes	**Estudos Retornados**	**1ª Seleção (Título, Resumo e Palavra-chave)**	**2ª Seleção (Introdução e Conclusão)**	
		Estudos Potencialmente Relevantes	**Excluídos**	**Incluídos**
IEEE Xplore	964	140	123	17
ACM	346	86	76	10
Scopus	1247	325	298	27
ScienceDirect	281	11	11	0
TOTAL	**2838**	**562**	**508**	**54**

Fonte: Elaborado pelo autor, com dados da revisão sistemática (2022)

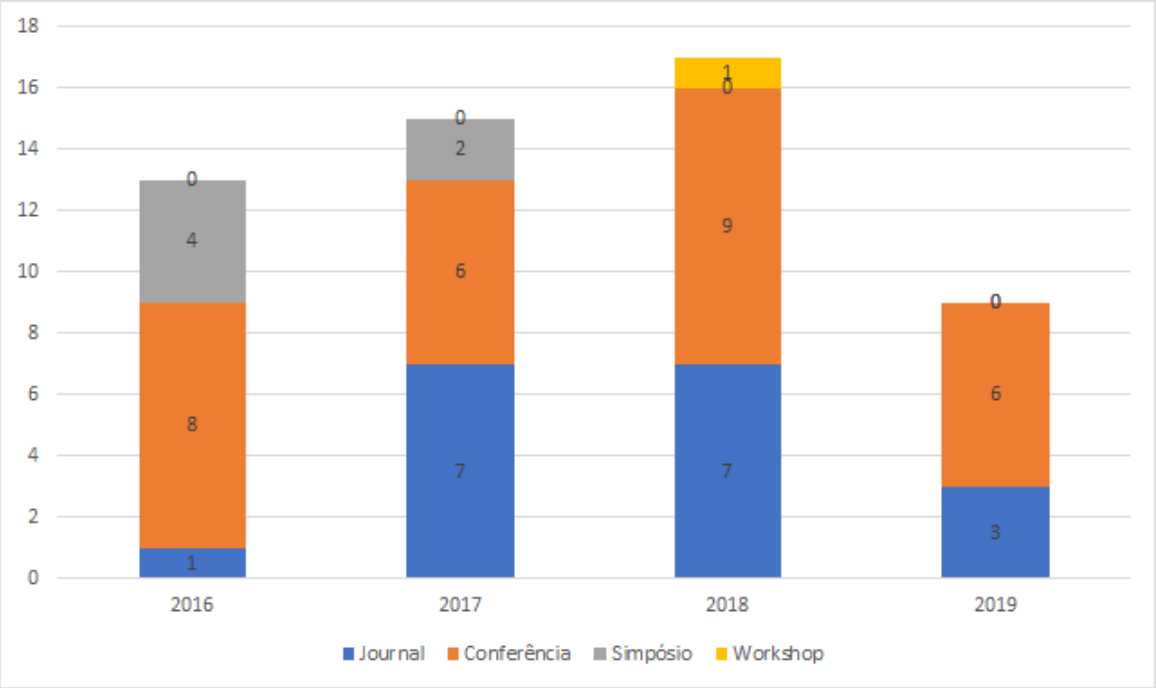
Com a leitura da introdução e conclusão dos estudos potencialmente relevantes, e utilizando-se os critérios de inclusão e exclusão, chegou-se a 54 estudos primários, disponíveis no Apêndice A deste trabalho. Assim, 510 trabalhos considerados potencialmente relevantes na primeira seleção foram excluídos e os principais motivos para exclusão foram: não respondiam a nenhuma questão de pesquisa; estudos que não estejam disponíveis livremente para consulta na web ou Portal da Capes; o foco principal do artigo ser avaliar uma nova tecnologia ou ferramenta em um caso da vida real; e, não apresentavam texto completo.

5.1.1 Distribuição dos Estudos

A partir dos estudos primários selecionados para esta revisão sistemática, realizamos uma análise temporal de publicação dos estudo afim de tentar identificar um possível interesse em pesquisas cientificas no contexto deste estudo.

A Figura 6 ilustra a distribuição dos estudos primários identificados pelo processo de seleção por tipo de estudo ao longo dos anos, onde é possível perceber que houve um numero crescente de publicações entre 2016 e 2018. Dos estudos desse período, 42,53% foram publicados em conferência e 27,77% em periódicos.

Figura 6 – Gráfico da distribuição dos estudos por tipo de publicação ao longo dos anos



Fonte: Elaborado pelo autor, com dados da revisão sistemática (2022)

Ao analisarmos o meio dos quais os estudos foram publicados, verificamos que 55,55% dos estudos foram publicados nos 4 periódicos apresentados no Quadro 9 de um total de 12 periódicos.

Quadro 9 – Estudos primários por periódico

Periódicos	Quantidade de Trabalhos
Empirical Software Engineering	2
IEEE Software	3
Information and Software Technology	2
Journal of Systems and Software	3
Total	10

Fonte: Elaborado pelo autor, com dados da revisão sistemática (2022)

Com relação aos estudos publicados em eventos, foram selecionados estudos de 27 eventos dentre os quais os 5 deles concentram 47,22%, os demais eventos obtiveram apenas 1 estudo em cada. Os eventos que tiveram mais estudos são apresentados no Quadro 10.

Dentre os 54 estudos primários da pesquisa, 69% se caracterizam como estudos experimentais (estudos baseados em evidências ou experimentos), 14% apresentam relato de experiência industrial, 13% são revisões sistemáticas da literatura, e 1% mapeamento sistemático da litera-

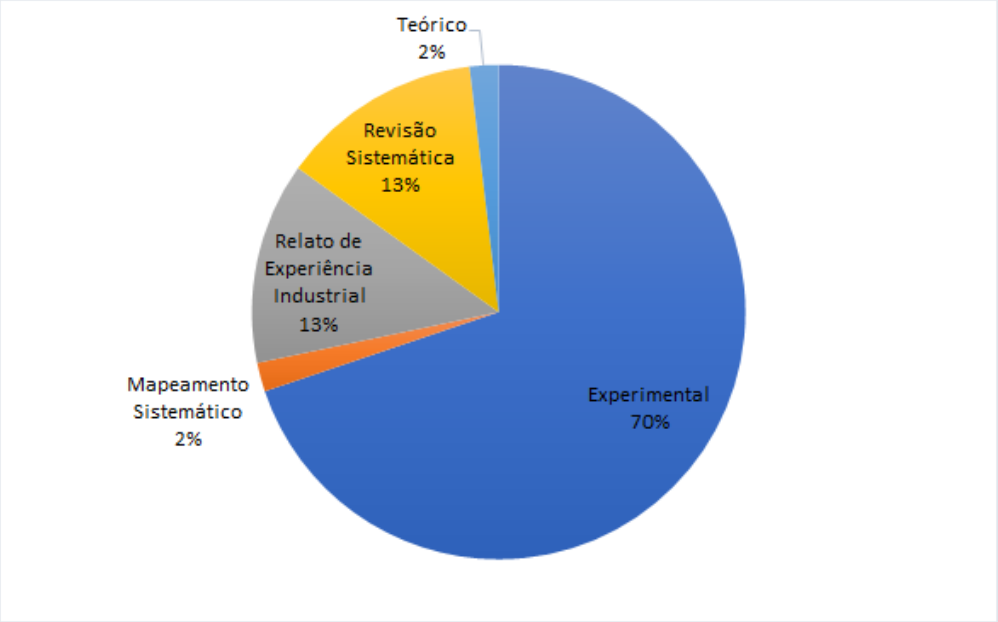
Quadro 10 – Estudos primários por eventos

Eventos	Quantidade de Trabalhos
ESEC/FSE - Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering	2
International Conference on Software Architecture (ICSA)	2
International Conference on Software Engineering	2
International Symposium on Empirical Software Engineering and Measurement (ESEM)	5
Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)	3
Total	17

Fonte: Elaborado pelo autor, com dados da revisão sistemática (2022)

tura e teóricos (estudos conceituais baseados em um entendimento de uma área, referenciando outros trabalhos relacionados), ou seja, apenas 1 estudo primário representa cada tipo de estudo. A Figura 7 ilustra a divisão dos tipos de estudos da pesquisa.

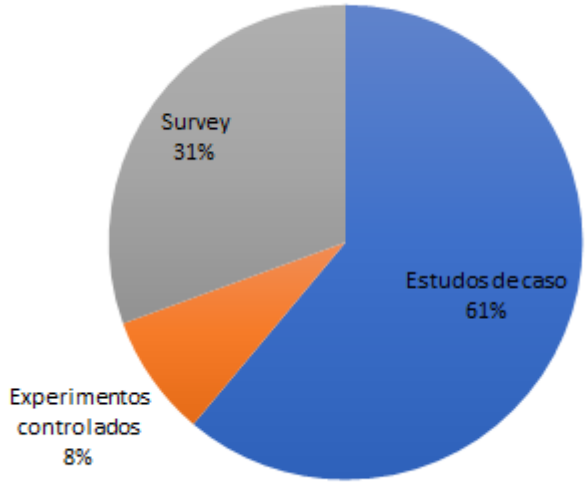
Figura 7 – Gráfico da distribuição dos trabalhos por tipo de estudo



Fonte: Elaborado pelo autor, com dados da revisão sistemática (2022)

Como visto, a maioria dos estudos primários foram do tipo experimental, e os principais métodos identificados na pesquisa para a realização do estudo, baseado na classificação de Easterbrook et al. (2008), foram: Estudo de Caso, Experimentos Controlados e Survey. A Figura 8 ilustra a distribuição dos métodos utilizados, no qual, a maioria, 42%, realizou um Estudo de Caso, 21% realizaram Survey e 6% conduziram Experimentos Controlados.

Figura 8 – Gráfico dos métodos empregados nos estudos do tipo experimental



Fonte: Elaborado pelo autor, com dados da revisão sistemática (2022)

5.1.2 Avaliação da Qualidade

Após a seleção dos estudos relevantes para a pequisa, a etapa de avaliação de qualidade dos mesmos foi realizada, conforme já apresentado no capítulo referente a metodologia desta pesquisa, as perguntas definidas para a avaliação foram conduzidas sobre cada estudo com o objetivo de determinar a credibilidade dos métodos utilizados e também dos resultados obtidos de cada estudo.

Na etapa de avaliação foi possível determinar o percentual da qualidade dos estudos através da somatória da pontuação obtida das respostas da avaliação de qualidade de cada artigo selecionado, sendo que a pontuação máxima que um estudo poderia alcançar de acordo com os critérios de qualidade definidos eram 40 pontos. A escala numérica foi baseada em Beecham et al. (2008), usando o intervalo de 1 até 5, e a cada valor representando um elemento, respectivamente, da seguinte classificação: Baixa, Média, Boa, Muito Boa e Excelente. Na Tabela 2 são apresentados os resultados da avaliação da qualidade.

Como pode ser observado, 88% dos trabalhos analisados apresentam qualidade acima da média de acordo com os critérios utilizados. É importante destacar que quanto melhor avaliado for um trabalho, maior deve ser a importância dada às evidências fornecidas por ele. A qualidade atribuída ao estudo será levada em consideração para a ordenação das evidências nos resultados das questões de pesquisa. Todos os valores da avaliação da qualidade se encontram

Tabela 2 – Qualidade dos estudos primários

	Baixa (<26%)	Média (26%-45%)	Boa (46%-65%)	Muito Boa (66%-85%)	Excelente (>86%)	**TOTAL**
Número de Estudos Primários	2	4	15	25	6	**54**
%	4%	8%	29%	48%	11%	**100%**

Fonte: Elaborado pelo autor, com dados da revisão sistemática, baseado em Beecham et al. (2008).

no Apêndice A, junto a lista de estudos primários.

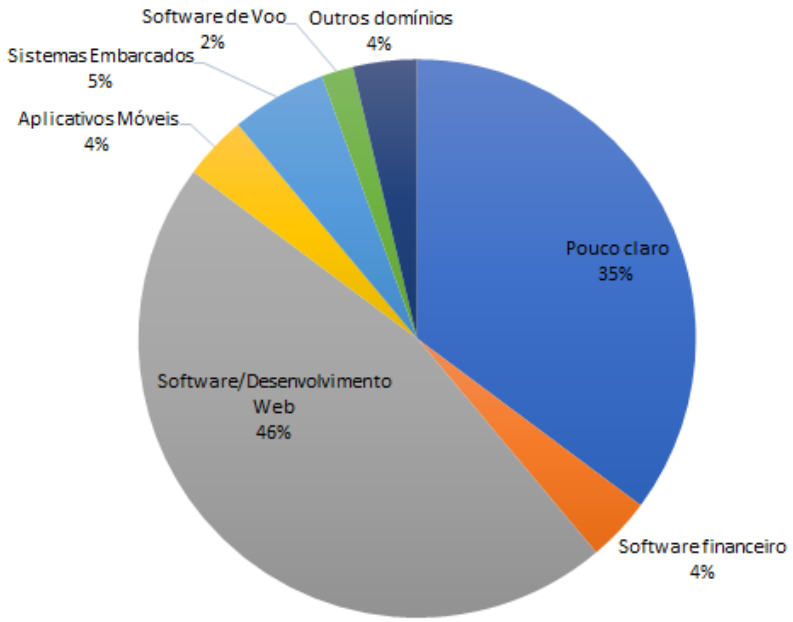
5.1.3 Domínio de aplicação

Outra análise que se pode fazer a fim de nos fornecer informações sobre em que tipo de projeto as ferramentas, desafios e práticas relatadas de CI/CD foram utilizadas, é quanto ao domínio da aplicação. A Figura 9 mostra os domínios de aplicação nos quais as abordagens, práticas, ferramentas, desafios e soluções podem ser utilizados.

Com relação ao domínio da aplicação, nem todos os artigos revisados forneceram essas informações, o que resultou na categorização de 35% dos estudos na categoria "pouco claro". Para os artigos que relataram os domínios de aplicativo, nós os classificamos em 6 domínios de aplicação. As ferramentas e práticas relatadas em um estudo podem ser aplicadas em mais de um domínio de aplicação com vários casos; por exemplo, a prática de testes de integração que foi relatada em estudos com domínio diferente, sendo eles "aplicativos móveis" (EP_20) e "software/desenvolvimento web" (EP_28). Se um estudo usar mais de um sistema como estudo de caso, contamos esse estudo N (número de sistemas) vezes na Figura 9.

Torna-se claro a partir da Figura 9 que o desenvolvimento web ganhou mais atenção para as práticas contínuas, seguido por sistemas embarcados, software financeiro. No entanto, existem 19 artigos sem qualquer informação sobre os tipos de projetos para os quais as abordagens, ferramentas e práticas contínuas propostas foram aplicadas.

Figura 9 – Gráfico da distribuição dos domínios de aplicação dos estudos primários



Fonte: Elaborado pelo autor, com dados da revisão sistemática (2022)

5.2 ANÁLISE DAS EVIDÊNCIAS

Nessa seção, são apresentados os resultados para cada questão de pesquisa. Na subseção 5.2.1 são apresentadas evidências quanto aos desafios relatados na adoção de práticas contínuas. Na subseção subseção 5.2.2 são apresentadas as evidências quanto as boas práticas para implementar com êxito práticas contínuas. Na subseção 5.2.3 são descritas as ferramentas que são utilizadas para projetar e implementar pipelines de implantação. Todas as evidências são devidamente referenciadas pelos 54 estudos, e os números das referências são precedidos por EP (Estudo Primário), como forma de deixar claras as referências da revisão sistemática. Como anteriormente citado, as informações quanto aos EP estão disponíveis no Apêndice A.

5.2.1 Q1. Quais desafios foram relatados para a adoção de práticas contínuas?

Esta questão buscou investigar os principais desafios em relação a adoção de um pipeline de implantação. Foram identificados pela pesquisa 32 desafios, agrupados em 9 categorias, sendo elas: Humano e Organizacional, Processo, Ferramentas, Design de Compilação, Integração, Arquitetura da Aplicação, Teste e Qualidade, Entrega, Infraestrutura e Monitoramento. As

evidências quanto a essas questões são descritas abaixo e sumarizados no Apêndice C.

5.2.1.1 Humano e Organizacional

Esta seção listará os desafios relacionados aos aspectos humanos e a estrutura organizacional de uma empresa, e, como podem afetar a implementação do CD. Os problemas mais relatados neste tema foram questões culturais, falta de comunicação e curva de aprendizado íngreme, falta de habilidades e desafios organizacionais.

D1. Cliente desmotivado

Para se adotar práticas de contínuas o cliente precisa estar envolvido ativamente na implantação contínua e automática. Shahin et al. (EP_14) dizem que embora as empresas pudessem fornecer entregas com a maior frequência possível as organizações de seus clientes, as culturas e políticas estabelecidas nelas, não suportavam a transição completa para a prática de CD, e portanto, as empresas tiveram que seguir intervalos de tempo predefinidos para lançar o software.

Seguem transcrições de evidências relacionadas a esse tópico:

- **EP_14** – *"We found that not all customers are mature enough to accept a continuous release."*

D2. Curva de aprendizado íngreme

Um pipeline de implantação bem estabelecido ajuda os novos membros a se familiarizarem rapidamente, no entanto, eventualmente, eles precisam aprender a atualizar o pipeline de implantação (EP_28). No caso de mudanças no pipeline, Zhang et al. (EP_36) afirmam que caso os desenvolvedores encontrem uma maior complexidade, maior latência e menor confiabilidade em fluxos de trabalho de CD anteriores, essas barreiras farão com que eles mudem para novos fluxos de trabalho de CD.

Seguem transcrições de evidências relacionadas a esse tópico:

- **EP_28** – *"High skills and knowledge demands. DevOps practices require that, in the least, the team as a whole has all necessary skills and knowledge to develop, integrate,*

test and deploy, which includes provisioning and upkeep target environments and infrastructure."

- **EP_36** – *"Steep learning curve. Complex processes and unfamiliar configurations make some developers abandon their old workflow. Like R76 told, their old workflow was "hard to learn, configure, or plain inefficient"."*

D3. Falta de habilidades

A equipe precisa ter todas as habilidades e conhecimentos necessários para desenvolver, integrar, testar e implantar, incluindo o provisionamento e manutenção de ambientes e infraestrutura de destino. Um pipeline de implantação bem estabelecido ajuda os novos membros a se familiarizarem rapidamente, pois esses, precisaram aprender sobre o pipeline (EP_28).

Seguem transcrições de evidências relacionadas a esse tópico:

- **EP_19** – *"Skills. The team needs to have a sufficient knowledge of what can be done and how to do it to be able to aim for higher CD maturity. Furthermore there is a need to have access to training to improve available skill set."*
- **EP_28** – *"DevOps practices require that, in the least, the team as a whole has all necessary skills and knowledge to develop, integrate, test and deploy, which includes provisioning and upkeep target environments and infrastructure."*

D4. Desafios organizacionais

As estruturas organizacionais podem ser desafios significativos na implementação da CD (EP_42). Uma organização distribuída pode ocasionar problemas de comunicação, dificultando a correção de falhas de compilação (EP_25). Chen (EP_42) declara que deve-se mudar o estrutura organizacional existente para criar implementações de sucesso.

Seguem transcrições de evidências relacionadas a esse tópico:

- **EP_25** – *"Distributed Organization. The case organization was distributed to several sites and countries. The participants thought that the communication was not working between the sites, especially when there were cultural differences. The communication problems were shown during the CD adoption by surprising breaking changes. These*

included API changes, version changes of libraries or changes in installation scripts. The breaking changes caused failing builds."

- **EP_42** – *"I mentioned organizational challenges in my earlier paper (Chen, 2015a). However, apart from the points discussed in Chen (2015a), I recently noticed another area in which evidence-based results are important. Organizational structures can be a significant barrier when implementing CD (Chen, 2015a)."*

D5. Falta de comunicação

A melhora da comunicação é considerado um dos benefícios de práticas continuas, no entanto, a falta dela dificulta a solução da tarefa ocasionando uma quebra no *build* (EP_33). Mårtensson et al. (EP_07) afirmam que a comunicação entre as equipes pode faltar, e Riungu-Kalliosaari et al. (EP_52) diz que se ela ocorrer apenas por meio de sistemas eletrônicos, causará atrasos nos tempos de reação aos problemas.

Seguem transcrições de evidências relacionadas a esse tópico:

- **EP_07** – *"One workshop participant from Company C described the presentation as "representative" and added "there are a lot of problems with communication between teams"."*
- **EP_52** – *"The respondents raised challenges related to communication patterns, organization cultures which are not malleable, different constraints stemming from the domain and environments, and the obscurity of the meaning of DevOps"*

D6. Dependências da equipe

A arquitetura de um aplicativo determina, em grande parte, a estrutura da equipe de desenvolvimento que trabalha nesse aplicativo (E_21). Shahin, Babar e Zhu (E_21) relatam ainda em seu estudo, que este desafio foi observado em equipes grandes que trabalhavam em uma base de código monolítica, no qual, para a implementação da aplicação, essas grandes equipes se dividiam em equipes menores.

As várias equipes trabalhando simultaneamente, geram uma dependência entre elas. Nessa situação pode haver equipes que tenham o controle do processo de implantação, que depen-

derão da liberação de outras equipes. A modificação desenvolvida por uma equipe pode causar falha de *build*.

Seguem transcrições de evidências relacionadas a esse tópico:

- **EP_21** – *"One of the challenges reported by the participants was that the dependent teams working on monolithic large-codebase were not successful in adopting and implement CD practice."*

D7. Falsa sensação de confiança

Outro desafio relatado é a falsa sensação de confiança, que trata-se de uma situação em que os desenvolvedores confiam cegamente nos testes. Essa falsa sensação pode levar um revisor de código a mesclar sem uma revisão completa.

Seguem transcrições de evidências relacionadas a esse tópico:

- **EP_33** – *"Another recurring problem is the false sense of confidence (25 occurrences). As opposed to the confidence benefit, respondents described the false sense of confidence as a situation where developers blindly trust the tests. One respondent synthesized this problem as follows"*

D8. Falta de conhecimento de domínio

A falta de conhecimento do domínio, pode impedir as contribuições de colaboradores novos, que sem orientação ou documentação adequada, não sabem como o projeto é organizado ou como começar a contribuir (EP_33). Pinto et al. (EP_33) afirmam que essa dificuldade é particularmente relevante para projetos de código aberto.

Seguem transcrições de evidências relacionadas a esse tópico:

- **EP_33** – *"Moreover, lack of domain knowledge, mentioned by 18 respondents, might hinder contributions from external contributors (eg, "lack of familiarity with a specific project")."*

D9. Questões Culturais

As questões culturais são vistas como um grande desafio na adoção de práticas de entrega contínua (EP_26), pois para que sejam adotadas tais práticas se faz necessário mudanças profundas na mentalidade. Essa mudança pode ser tornar um desafio maior se a cultura organizacional for arraigada (EP_52). Riungu-Kalliosaari et al. (EP_52) citam como exemplo o caso dos desenvolvedores que precisam assumir tarefas com as quais não estão acostumados e podem ter reservas em aceitar novas responsabilidades.

Seguem transcrições de evidências relacionadas a esse tópico:

- **EP_26** – *"The principles and practices of continuous delivery can be implemented in all kinds of environments, from mainframes to firmware to those that are highly regulated, but it's certainly not easy.[...] Typically, the biggest obstacles to this transformation are organizational culture and architecture."*
- **EP_52** – *"DevOps adoption also highlights cultural matters. Profound changes to the cultural mindset are required and the deep-seated company culture can be a challenge.[...] The cultural aspects are significant, as has been previously stated [7,13]. The size of the company or having company-wide support for the change might matter. Smaller companies are in a better position to react faster to changes."*

5.2.1.2 Processo

Esta seção listará os desafios diretamente ou indiretamente relacionados ao processo de desenvolvimento da equipe que deseja implementar práticas contínuas. Mesmo que essas práticas não estejam diretamente relacionadas a um processo de desenvolvimento específico, foram encontrados diferentes aspectos do processo que podem tornar mais difícil a adoção das práticas contínuas com sucesso.

D10. Ambientes heterogêneos

Em ambientes heterogêneos há uma dificuldade na replicação dos serviços em produção para o ambiente para verificação e teste, pois tais serviços podem ser complexos o suficiente para dificultar tal replicação, tornando o teste automatizado se torna menos confiável.

Seguem transcrições de evidências relacionadas a esse tópico:

- **EP_21** – *"Our analysis of the data revealed that it was challenging for a couple of practitioners to design applications for different operations environments, in which they may have had difficulty to make consistency in a set of heterogeneous operations environments in order to seamlessly transfer and deploy the application in all production environments."*
- **EP_52** – *"As a consequence, automated testing becomes less trustworthy meaning that heterogeneous environments provide a challenge for successful DevOps adoption."*

D11. Múltiplos Ambientes

Em um determinado aplicativo, pode haver a necessidade de se tem que ter vários ambientes por inúmeras razões, como exemplo de ambiente tem-se, local, desenvolvimento, *staging* e produção. Nesse caso, ter vários ambientes também pode ser um problema, pois se a compilação falhar, poderá ser difícil depurá-la, especialmente se o problema não ocorrer no ambiente local.

Seguem transcrições de evidências relacionadas a esse tópico:

- **EP_33** – *"Interestingly, one respondent drew attention to the fact that the benefit multiple environments can also be a problem, [...]"*

5.2.1.3 Ferramentas

Nesta seção, será abordado os desafios associados às ferramentas usadas durante a implementação da integração e entrega contínua.

D12. Ferramentas em constante mudança

Alguns profissionais acabam obtendo muito conhecimento em relação a um determinado tipo de ambiente e provavelmente projetando sistemas com especificidades para esse determinado tipo de ambiente. Se o ambiente mudar as decisões de design específicas não funcionaram. A influência de ferramentas em constante mudança no design de arquitetura para permitir a prática do CD acaba se tornando um desafio (EP_21).

Seguem transcrições de evidências relacionadas a esse tópico:

- **EP_21** – *"Another challenge reported at architectural level was the influence of ever-changing environments and tools on architecture design to enable CD practice."*

D13. Falta de suporte de ferramenta

Quando uma equipe o planjamento de adoção de um pipeline CI/CD deve-se tomar muito cuidado com a escolha de ferramentas, pois uma determinada tarefa pode não atender ao que se esperava, podendo ocasionar um retrabalho para migrar para outra ferramenta que atenda melhor. Debroy, Miller e Brimble (EP_35) em seu estudo por ainda não terem a licença do Google Cloud basearam-se na integração do Visual Studio Team Services e Azure, no entanto, essa escolha de ferramenta impossibilitou-os de registrar os container no com Google Container Registry, causando um atraso na adoção do pipeline CI/CD.

Seguem transcrições de evidências relacionadas a esse tópico:

- **EP_35** – *"At this point, it looked like due to lack of tool support – we were either going to have to give up Google Cloud as an option or create entirely new CI/CD pipelines to support it."*

5.2.1.4 Design de Compilação

Esta seção abordará os desafios causados por decisões de design de compilação.

D14. Longos tempos de espera para builds

Uma compilação demorada reflete no modo de trabalho da equipe de desenvolvimento, afetando a eficiência, fazendo com que alguns desenvolvedores mudem seus fluxos de trabalho. Debroy, Miller e Brimble (EP_35) relatam o problema que tiveram em aplicação monolitica que estava migrando para arquitetura microsserviços, na qual, a construção dos microsserviços era realizada pelo pipeline CI/CD. Como haviam muitos microsserviços distribuidos entre várias equipes os *builds* entraram numa fila, gerando atrasos e muito tempo perdido.

Seguem transcrições de evidências relacionadas a esse tópico:

- **EP_35** – *"Long Wait Times for Builds/Releases: At least one of the problems becomes evident when we directly apply the existing CI/CD setup as-is to support the desired goals[...]"*
- **EP_36** – *"Overly long build times. As we found earlier when asking about CD needs, build speed affects the developers' work efficiency."*

5.2.1.5 Integração

Esta seção listará os desafios que surgem quando o código é integrado à ramificação principal do controle de versão.

D15. Múltiplos Branches

O uso de múltiplos *branches* no controle de versão atrasa a integração do novo código a ramificação principal, fazendo com que o desenvolvimento se torne complexo, pois equipes diferentes trabalhavam em ramos diferentes, gerando uma dificuldade na comunicação, além de que as correções de *bugs* e até mesmo novos recursos às vezes tinham que ser aplicados a várias ramificações, exigindo um esforço adicional (EP_25).

Seguem transcrições de evidências relacionadas a esse tópico:

- **EP_05** – *"Foi identificado que alguns sistemas possuíam muitas ramificações, o que dificultava a junção do código (merge) dos diversos ramos de desenvolvimento do sistema, quando era necessário gerar uma nova versão para ser testada."*
- **EP_25** – *"In addition, working with multiple branches was described to cause complexity during development."*

D16. Falhas de Build

As falhas de construção não necessariamente são um problema para a prática de CD, no entanto, se tornarão problemáticas se forem comuns e não forem corrigidas imediatamente (EP_25). Pinto et al. (EP_33) relataram diversas causas para as falhas de construção, como por exemplo, testes inadequados, mudanças de versão da linguagem, complexidade da base de código, entre outros.

Seguem transcrições de evidências relacionadas a esse tópico:

- **EP_25** – *"Failing builds: CD pipeline builds were often failing and not fixed immediately afterwards."*
- **EP_33** – *"We found several technical reasons that might explain build breakage. Among the most common ones, there is inadequate testing[...]"*

D17. Branches longas

Trabalhar em ramificações ou *branches* longas vai de encontro com a filosofia da integração contínua, pois a integração da nova funcionalidade é adiada, ocasionando grandes problema de integração. Mårtensso, Hammarströme e Bosch (EP_03) relataram esse desafio a prática GitFlow, informando que os desenvolvedores preocupam-se em integra constantemente na *branch* de desenvolvimento do que na principal, fazendo com que as entregas na *branch* principal não possuam valor.

Seguem transcrições de evidências relacionadas a esse tópico:

- **EP_03** – *"The third factor "no evident value in delivering often to the mainline" (shown in Fig. 7) include comments from only five interviewees. However, we believe that the interview responses on previous questions, describing a way of working with long-lived branches and delivering with a low frequency to the mainline supports "no evident value" as an important factor."*
- **EP_07** – *"The product's architecture: During the interviews, we identified problems related to teams working on a branch too long, and then having huge problems when they try to integrate (described in described Section IV-B)."*

5.2.1.6 Arquitetura da Aplicação

Nesta seção, será examinado os desafios relacionados à arquitetura do aplicação. Alguns tipos de arquitetura podem tornar muito difícil a implementação do CD.

D18. Dependências

O gerenciamento de dependências e o controle de versão são atividades principais dos sistemas de *build*, no entanto, às vezes os desenvolvedores não atualizam as dependências, logo o servidor de CI detecta erros que não aconteceriam localmente. Pinto *et al.l* (EP_33) dizem que a maioria dos erros de compilação se deve à falta de dependências.

A dependência de aplicações com outros sistemas podem inibir as organizações a mudar de entrega contínua para implantação contínua, pois isso significa que elas precisariam garantir que não haja nenhum problema de integração ao implantar um aplicativo para produção (EP_14).

Seguem transcrições de evidências relacionadas a esse tópico:

- **EP_14** – *"Our study has found that albeit an application might be at deployable state, dependencies between that application and other systems may have inhibited some of the participants' organizations to transition from CDE to CD."*
- **EP_33** – *"It is important to note that dependency management and version changes are two key activities of build systems. Therefore, developers that may face technical problems with the usage of build systems may also face those with CI systems."*

D19. Arquitetura Altamente Acoplada

A arquitetura altamente acoplada é um desafio que está relacionado com a adoção de práticas de implantação contínua em aplicativos monolíticos pelas organizações. Esse tipo de sistema possui muitas dependências multifuncionais, nas quais cada dependência precisa ser gerenciada no pipeline de CD para implantação de software, portanto se um componente precisar ser implantado no ambiente de produção, espera-se que todas as dependências sejam implantadas juntamente com ele (EP_21).

Os aplicativos podem ser monolíticos do ponto de vista do banco de dados, ou seja, a aplicação ter uma arquitetura de microsserviços e usar um único banco de dados ao invés de cada serviço possuir seu próprio banco. Nesse caso, o banco de dados pode criar gargalos operacionais e gradualmente se torna uma unidade não implantável, pois a alteração de qualquer funcionalidade no aplicativo pode ocasionar mudanças no esquema de banco de dados também (EP_21).

Seguem transcrições de evidências relacionadas a esse tópico:

- **EP_21** – *"Our analysis has revealed that highly coupled architecture presents the biggest challenge when organizations are going to move to continuous deployment practice."*
- **EP_25** – *"Unsuitable Architecture. The participants reported that the product architecture did not support the adoption of CD.[...] Other than that, the architecture required the whole product to be released as a whole."*

D20. Restrições de Domínio

As restrições de domínio fazem com surjam muitos desafios na aplicabilidade do CD, podendo alterar a frequência de liberação de software (mudanças) para produção. Embora a prática de entrega contínua seja facilmente aplicada a aplicativos baseados na web, ela pode se tornar difícil se aplicada a outros domínios, como sistemas embarcados e sistemas financeiros (EP_14 apud Skelton e O'Dell, 2016).

Seguem transcrições de evidências relacionadas a esse tópico:

- **EP_14** – *"Many challenges in the applicability of CD arise due to domain constraints."*

5.2.1.7 Teste

Nesta seção, serão discutidos os desafios relacionados aos testes de software na adoção de um pipeline CI/CD. A parte central desse pipeline envolve os testes automatizados, sendo o conjunto de testes executado toda vez que há uma mudança na base de código, para validar que a mudança pode ir para a produção.

D21. Baixa Cobertura de Teste

O baixo nível de cobertura de testes reduz a confiança das organizações na possibilidade de suas aplicações estarem prontas para serem implantadas em produção (EP_14, EP_25). Essa baixa cobertura se torna um desafio às práticas CD, pois ao praticá-las deve-se estar confiante para lançar o produto após a execução dos testes.

Seguem transcrições de evidências relacionadas a esse tópico:

- **EP_14** – *"Lack of sufficient automated test coverage was also deemed as a bottleneck to transition from CDE to CD [...]."*
- **EP_25** – *"The identified direct signs of a dysfunctional CD practice were failing builds, flaky tests, low test coverage and slow feedback. "*

D22. Falta de Teste de Aceitação Automatizado

Os testes de aceitação tornam a construção do conjunto de teste mais complexo e exigindo mais esforço para configurar e gerenciar testes automatizados. Shahin et al. (EP_14) relatou a preocupação que os desenvolvedores tem sobre os benefícios potenciais de automatizar o teste de aceitação em comparação com suas complexidades e custos associados, elencando esta preocupação como um dos principais motivos pelos quais esse tipo de teste não foram totalmente automatizados.

Seguem transcrições de evidências relacionadas a esse tópico:

- **EP_14** – *"The interviewees disclosed that they considerably succeeded in automating unit and integration tests, but automating the tests occurring at the end of development process such as (user) acceptance test and performance test has remained a challenge and requires heavy workloads and time."*

D23. Testes de Longa Duração

O feedback rápido é um dos benefícios da integração contínua, portanto possuir testes de longa duração acarretará em um atraso do feedback dos desenvolvedores, consequentemente, levará a um aumento do tempo do ciclo de entrega.

Seguem transcrições de evidências relacionadas a esse tópico:

- **EP_14** – *"Long running tests not only increased the cycle time (i.e., the time required to get code from code repository into production) in a CDP, but also have hindered developers to getting real-time feedback."*

D24. Falta de Testes

Os testes automatizados garantem que o código produzido não irá introduzir nenhum defeito ao sistema, além de aumentar a confiança da equipe no produto de software. A falta de testes torna-se um desafio, devido a que a execução de testes automatizados é uma das práticas chaves para que a equipe possa colher os benefícios do CI.

Há diversos fatores que influenciam a falta de testes em um projeto. Pinto et al. (EP_33), em seu estudo, elenca a falta de cultura de testes como um motivo social associado à quebra de construção, devido ao fato de que os desenvolvedores não executam os testes e não constroem em suas máquinas antes de enviarem as mudanças, delegando a execução dos testes somente para o servidor CI. Já Laukkanen et al. (EP_32 apud Paternoster et al., 2014; Giardino et al., 2016) elencou a falta de testes como uma das características da engenharia de lançamento das *startups*, devido a falta de recusos.

Seguem transcrições de evidências relacionadas a esse tópico:

- **EP_17** – *"Microservices introduces testing challenges, which mainly arise from changes that impact interactions among services."*
- **EP_33** – *"In our study, we perceived that there is a lack of testing culture, which might motivate educators to place additional care in providing testing courses."*

D25. Testes Instáveis

Flaky test ou testes instáveis são testes que podem falhar aleatoriamente na mesma configuração ou mesmo quando não há problemas na alteração do código, por este motivo torna-ser difícil de reproduzir a falha, dificultando a confiança nos resultados do *build* e aumentando o esforço de manutenção da construção.

Seguem transcrições de evidências relacionadas a esse tópico:

- **EP_25** – *"Some of the tests used in the CD pipeline were flaky and could fail randomly even if a software change did not have any problems in it. This makes it difficult to trust the build results and increases the build maintenance effort."*

- **EP_33** – *"Moreover, seven respondents reported that flaky tests may affect only few jobs. A flaky test is a test that could fail or pass for the same configuration and, therefore, can create several problems since it can be hard to reproduce. "*

D26. Falta de Estratégia de Teste

A estratégia de teste defini os estágios e os tipos de teste que devem ser usados e quando usá-los. A falta dessa estratégia faz com que sejam produzidos testes redundantes e funcionalidades sejam testadas em nível errado, gerando lentidão na execução pipeline (EP_25).

Seguem transcrições de evidências relacionadas a esse tópico:

- **EP_25** – *"The lack of a testing strategy had caused duplicate testing in the CD pipeline and the pipeline was considered slow because of that."*

D27. Teste Inadequado

A construção de testes inadequados é um das motivos mais comuns de uma falha de construção, devido a falta de uma cultura de teste, o que leva os desenvolvedores a pular ou escrever testes ruins e/ou ingênuos (EP_33).

Seguem transcrições de evidências relacionadas a esse tópico:

- **EP_33** – *"Inadequate testing and time pressure are the most common technical and social reasons related to build breakage."*

5.2.1.8 Entrega

Esta seção listará os desafios relacionados a entrega e implantação de software. Os problemas relacionados a entrega foram relatados em três artigos, sendo eles, a falta de um mecanismo de reversão (EP_14) e a demora do processo de entrega (EP_03, EP_35).

D28. Falta de Mecanismo de Reversão

A falta de um mecanismo de reversão eficiente força as empresas a reduzir a frequência de lançamento de versões em produção, além de gerar risco de entregar código com erros a seus

clientes (EP_14).

Seguem transcrições de evidências relacionadas a esse tópico:

- **EP_14** – *" Lack of efficient and automated rollback mechanism to quickly recover issues in deployment process may put software organizations at risk of delivering buggy code to their customers. "*

D29. Processo de Entrega Demorado

A demora do processo de entrega acaba frustrando os desenvolvedores devido a longa espera e *feedback* lento (EP_35). O principal fator que causa demora no processo de entrega são as atividades de testes, especialmente as atividades de teste manual, ou atividades em que o desenvolvedor deve estar de prontidão para realizar as etapas manuais para interpretar os resultados do teste ou para passar de uma atividade de teste para outra (EP_03). Em uma arquitetura de microsserviços essa demora pode se tornar um gargalo, se houver poucas instâncias do servidor CI e muitas versões de serviços diferentes a serem lançadas, quando as instâncias estiverem ocupadas as novas solicitações serão inseridas uma fila para serem executadas posteriormente.

Seguem transcrições de evidências relacionadas a esse tópico:

- **EP_35** – *"At the same time the problem was severe in terms of wall-clock time lost just waiting for a build/release."*

5.2.1.9 Infraestrutura

Nesta seção, serão examinadas as dificuldades relacionadas à infraestrutura necessária para implementar práticas contínuas.

D30. Falta de Recursos

A necessidade de recursos pode ser um bloqueador para empresas menores devido ao seu custo. Laukkanen et. al (EP_32) exemplificam o desafio enfrentado por uma *startup* que por causa da falta de recursos, não tinha nenhum ambiente de teste semelhante ao de produção para verificação interna do produto.

A falta de recursos pode levar a uma competição devido as equipes precisarem compartilhar a infraestrutura de compilação, caso a capacidade computacional suficiente para suprimir a demanda toda a empresa (EP_35).

Seguem transcrições de evidências relacionadas a esse tópico:

- **EP_32** – *"The lack of resources was visible in that the organization did not have any production-like test environment for internal verification of the product."*
- **EP_35** – *"If both agents were busy with a current build or release, then any more requests would just get queued, and the queue itself would start growing."*

D31. Configuração Manual de Software

A configuração manual da aplicação em produção representa um desafio para o sucesso do CD, devido a ser processo complexo e sujeito a erros de configuração. Shahin et al. (EP_14) elenca dois outros motivos para se ter configuração manual e provisionamento: (i) falta de ferramentas maduras; e (ii) não tem muito valor em automatizar a configuração e o provisionamento.

Seguem transcrições de evidências relacionadas a esse tópico:

- **EP_14** – *"[...] several of the interviews' and survey participants have said that manual configuration of complex software, particularly when there is a tight coupling between software and hardware, and regulatory environments represented a significant obstacle to CD success."*

D32. Esforço Inicial

No início da adoção um um esforço inicial é necessário para configurar o sistema de CD e monitorá-lo. De acordo com (LAUKKANEN; ITKONEN; LASSENIUS, 2017), o esforço inicial percebido para implementar o sistema de CI pode causar uma situação em que se torna difícil motivar as partes interessadas para a adoção das práticas CI/CD.

Seguem transcrições de evidências relacionadas a esse tópico:

- **EP_05** – *"No início da adoção do processo é necessário um esforço inicial para configurar as ferramentas, treinar a equipe e implantar os subprocesso, o que irá gerar um grande*

impacto na velocidade do desenvolvimento e manutenções durante o início da adoção do processo."

5.2.2 Q2. Que práticas foram relatadas para implementar com êxito práticas contínuas?

Esta questão buscou encontrar boas práticas adotadas para implementar práticas contínuas. A partir das evidências coletadas pela revisão sistemática, 20 Melhores práticas a serem adotadas no pipeline de implantação foram identificadas. Essas práticas são descritas abaixo e sumarizadas pelo Apêndice D.

MP1. Adoção de Práticas Ágeis

As práticas ágeis mantêm as equipes em um ritmo constante de produção e de entrega das funcionalidades e são vistas como boas para incentivar a adoção de integração e entrega contínua. Vassallo et al. (EP_06) declaram que a introdução de um pipeline de CD impõe um processo de desenvolvimento ágil para reduzir o esforço e a duração de teste e implantação. Dentre as várias práticas existentes, foram mencionados os métodos Scrum (EP_06, EP_28, EP_45, EP_49), Kanban (EP_28, EP_42, EP_45, EP_49), *Extreme Programming* (XP) (EP_25, EP_45) e Lean (EP_28, EP_45).

Seguem transcrições de evidências relacionadas a esse tópico:

- **EP_06** – *"The next step was the introduction of a CD pipeline enforcing an agile development process to reduce the testing and deployment effort and duration, especially because such activities were used to be mainly manual work for two separate teams."*
- **EP_28** – *"Agile, lean practices (and their adaptations): No explicit agile method in cases, except for Case B, which was using Scrum. However, implicit adaptation to agile Scrum and lean Kanban. From Scrum practices, e.g., daily/weekly stand-up meetings, retrospectives and task estimation. From lean, task tracking and a Kanban board were used in all cases. Rationale for not using Scrum in its entirety was to be more efficient and less constrained to the procedures.[...] Our findings show toolchain use and support for the activities of the deployment pipeline in all cases, while also applying the principles and practices of trunk-based development, code reviews, CI and Agile and lean."*

MP2. Padronizar o Fluxo de trabalho no Sistema de Controle de Versões

Ter o código fonte versionado é uma das primícias das práticas contínuas, no entanto, adotar a prática de versionamento sem tem ter um fluxo de trabalho definido pode dificultar a implantação contínua. Lwakatare et al. (EP_28), Ivanov e Smolander (EP_29) sugeriram em seus estudos a padronização do fluxo de trabalho dos sistemas de controle de versão utilizando o *Trunk-Based Development* (TBD) e o *Git Flow*, respectivamente.

Git Flow ou Fluxo de trabalho Git, um modelo de organização de branches criado por Vincent Driessen[1] que sugere que sejam criadas 4 ramificações além da principal sendo elas: (i) *master*: contém o código em nível de produção; (ii) *develop*: contém o código em nível preparatório para o próximo implantação, ou seja, quando as novas funcionalidades são terminadas, elas são juntadas com esta *branch*; (iii) *feature branches*: são branches criadas pra desenvolver um novo recurso; (iv) *release branches*: *branch* com um nível de confiança maior do que a *branch develop*, se encontrando em nível de preparação para ser mesclada com a *branch master*, caso tenha ocorrido alguma correção de bug nesta *branch*, com a *develop*; e (v) *hotfixes*: branches no qual são realizadas correções de bugs críticos encontrados em ambiente de produção, e que por isso são criadas a partir da branch master, e são juntadas diretamente com a branch master e com a *branch develop* (DRIESSEN, 2010).

Driessen (2010) adicionou uma nota em seu artigo em que sugere adotar um fluxo de trabalho mais simples do que o seu, para aplicativos web e equipes está adotaram entrega contínua de software.

Seguem transcrições de evidências relacionadas a esse tópico:

- **EP_39** – *"The model uses so-called feature branches. There are a develop branch with the latest development changes merged from feature branches, release branches with the version ready for internal testing and a master branch with a stable production-ready version.[...] The Git workflow model suggests that the developer checks out the code and creates a feature branch. The input to the pipeline is a commit to the feature branch. After the local development and testing, the changes from the branch are committed into the repository.[...] The reason for this is that commits into the release branch are done less often than to the develop branch. It is assumed that the develop branch will receive five merge requests per day and the release branch only one merge request per*

[1] https://nvie.com/posts/a-successful-git-branching-model/

two weeks. The changes in the release branch are deployed for internal testing conducted by the end users within the company."

Trunk-based development (TBD) ou em português Desenvolvimento baseando em troncos, é um modelo de ramificação do sistema de controle de versão, onde os desenvolvedores colaboram em código em um único *branch* chamado "tronco" (*master* no Git), resistindo a qualquer pressão para criar ramificações de desenvolvimento de longa duração (HAMMANT; SMITH, 2017).

Esse fluxo de trabalho facilita a adoção de práticas contínuas, como integração e entrega contínua, pois diferentemente de outros fluxos que adiam a integração da *feature* até que esteja completa, orientando a construção de pequenos lotes de códigos que devem ser integrados frequentemente para eliminar integrações longas (EP_28).

Seguem transcrições de evidências relacionadas a esse tópico:

- **EP_28** – *"Trunk-based development: Frequent merges of branches to the mainline in all cases to minimise effort and conflicts resulting from code merges. In Case D, feature branches were released and thus maintained for some APIs. [...] Our findings show toolchain use and support for the activities of the deployment pipeline in all cases, while also applying the principles and practices of trunk-based development, code reviews, CI and Agile and lean."*

MP3. Mudanças do Esquema do Banco de Dados no Sistema de Controle de Versão

Quando se desenvolve software com um banco de dados SQL, as novas funcionalidades podem exigir que sejam realizadas alterações no banco de dados. Essas alterações pode ser adicionar uma nova coluna ou correções de dados. Ivanov e Smolander (EP_39) classificou a prática de versionar as mudanças do esquema de banco de dados como uma das 25 praticas importantes de 27 práticas sugeridas de automação DevOps.

Seguem transcrições de evidências relacionadas a esse tópico:

- **EP_39** – *""Version control DB [database] schema changes" (#27) was called as an important practice for relational databases but in the context of the case project and its schemaless database this practice was also marked as not important."*

MP4. Todos os commits estarem vinculados às tarefas

Toda a mudança no código deve estar relacionada a alguma tarefa, tornado as mudanças parte de um plano geral. Além de gerar rastreabilidade nas mudanças que podem ser usadas posteriormente em uma auditoria ou documentação, permitindo o acompanhamento do ritmo da equipe. Ivanov e Smolander (EP_39) categorizaram essa prática como sendo relacionada ao sistema de controle de versão e como uma das 27 práticas sugeridas de automação DevOps.

MP5. Commit de código com mais frequência

A prática de "commitar frequentemente" alinhada com a prática de "commits pequenos" contribuem diretamente para o cerne do integração contínua que é o feedback rápido, pois ao fazer commits menores e com mais frequência, é reduzido o esforço de depuração e os programadores podem resolver mais bugs e problemas (EP_27, EP_34). Rahman et al. (EP_34) afirmam que apenas a adoção de CI pode não ser suficiente para colher os benefícios de uso e nem melhorar o processo de desenvolvimento de software, pois para que isso aconteça se faz necessário que os desenvolvedores adotem as melhores práticas de CI, como por exemplo, a prática de commits frequentes.

Seguem transcrições de evidências relacionadas a esse tópico:

- **EP_27** – *"Continuous integration encourages developers to "break down their work into small chunks of a few hours each", as smaller and more frequent commits helps them keep track of their progress and reduces the debugging effort [15], [16]."*
- **EP_31** – *"[...] the team has identify the practice of "commit often, commit little"as important - they commit small code increments (e.g. 10-20 lines of code) and do that very often."*

MP6. Commits pequenos

Como mencionado na anteriormente as prática de "commits frequentes" e "commits pequenos" favorecem que os desenvolvedores colham os beneficios de adotar CI, como por exemplo. o feedback rápido (EP_34). Zhao et al. (EP_27) afirmam que com a diminuição da quanti-

dade de códigos do commit e o aumento de sua frequência, facilita o processo de verificação do build, bem como a prática de revisão de código através do *pull request*.

Seguem transcrições de evidências relacionadas a esse tópico:

- **EP_31** – *"[...] the team has identify the practice of "commit often, commit little"as important - they commit small code increments (e.g. 10-20 lines of code) and do that very often."*
- **EP_37** – *"The authors [Ståhl e Bosch (2014) e Zhao et al. (2017)] observe that practices such as "commit often" and "commit small" are indeed employed after the adoption of CI. However, the growing trend of closed issues slow down after the adoption of CI."*

MP7. Trabalhar em Pequenos lotes

Para que a equipe de desenvolvimento tenha uma rápida progressão da etapa de desenvolvimento para a produção, por meio de testes e operações se faz necessário trabalhar com código em pequenos lotes.

O lote é a unidade de trabalho que se move através do fluxo de trabalho ou fluxo de valor. Quanto menor um lote de trabalho mais rápido chegará à próxima etapa ou será entregue ao cliente, reduzindo o tempo de ciclo e fazendo os desenvolvedores obterem feedback mais rápido, pois o código produzido é revisado em cada etapa.

Seguem transcrições de evidências relacionadas a esse tópico:

- **EP_15** – *"[...] encouraged DevOps practices such as fast feedback, small batch sizes, and independent releases, emphasizing increased team autonomy"*

MP8. Refatoração

Refatoração, segundo Fowler (2018), "é o processo de mudar um sistema de software de uma forma que não altere o comportamento externo do código, mas melhore sua estrutura interna." As principais razões para refatorar incluem melhorar a compreensão do programa, permitir fazer alterações mais fáceis e ajudar a encontrar bugs. A refatoração pode ser realizada em conjunto com outras tarefas ou de forma separada das demais tarefas.

O estudo indica que as tarefas de refatoração são geralmente programadas de maneira adequada, os que não o fazem alegam que consome muito tempo, reduz o esforço de tarefas de implementação de recursos ou que não perceberam claramente as vantagens da refatoração.

Seguem transcrições de evidências relacionadas a esse tópico:

- **EP_06** – *"[...] refactoring is usually properly scheduled. The main reasons for refactoring include improving program comprehension (87%), allow making changes easier (77%), and help to find bugs (24%).Those who not schedule refactoring tasks, they do it either because they are too time consuming and take effort away from feature implementation tasks (27%), or because they do not clearly perceive refactoring advantages (9%).[...] Differently from what Fowler reported [25], refactoring tasks are often performed together with other tasks[...]"*

MP9. Revisão de Código

A revisão de código tem objetivos bem claros detectar *bad smells* e encontrar defeitos, compartilhar o conhecimento do código, os estilos e padrões de codificação, encontrar maneiras alternativas de implementar um recurso, garantir a qualidade. Os fatores considerados importantes na revisão do código incluem o conhecimento do revisor, o tamanho das alterações no código a serem revisadas e o suporte do ambiente de revisão.

A revisão de código geralmente é realizada através de *pull request* pra mesclar um *branch* de trabalho ao *branch master*. Nesse momento, os revisores do código verificam as mudanças, dão feedback crítico e sugeriram se as mudanças deveriam ou não ser aceitas para o produto. O desenvolvedor pode melhorar as mudanças no *branch* com base no feedback e até que seja corrigido, os revisores não o aprovam. O *brach* de trabalho também foi continuamente testado pelo sistema de CI e não seria integrado ao produto se todos os testes não fossem aprovados (EP_32).

Seguem transcrições de evidências relacionadas a esse tópico:

- **EP_28** – *"Main purpose was to ensure quality and an opportunity to share information about coding styles and standards. Factors in code review considered important included knowledge of the reviewer, size of code changes to be reviewed and review environment support."*

- **EP_50** – *"Code reviews are prevalent in continuous deployment processes. Because developers are fully responsible for the entire lifecycle of the software, code reviews are taken more seriously and there is far less resistance to them."*

A ***peer-review*** ou revisão por pares é um tipo de revisão de software em que o código produzido é revisado por outro desenvolvedor, com a finalidade de detectar e corrigir defeitos além de avaliar a qualidade. Ivanov e Smolander (EP_39) listaram a revisão por pares como uma sugerida de automação DevOps, classificando-a como teste e controle de qualidade.

MP10. Análise Estática de Código Automática

A análise estática automática complementa o CI, pois o uso de ferramentas que fazem essa análise auxiliam os desenvolvedores a testarem o seu código sem realmente executá-lo, encontrando falhas de programação, vulnerabilidades, *bad smells* (indicador de um possível problema estrutural em código-fonte, que pode ser melhorado via refatoração (FOWLER, 2018)), verificando se os desenvolvedores estão cumprindo as diretrizes de codificação e etc, que poderiam ir para o cliente.

Com o uso de ferramentas automatizadas de análise de código estático no CI, esses problemas encontrados são encontrados antes de a versão ser lançada. Essas ferramentas produziriam avisos que todos veriam e em alguns casos produzindo quebras de *build*. E ao mesmo tempo, no entanto, tal efeito sugere que o uso dessas ferramentas com prudência ajuda a evitar muitas quebras de compilação desnecessárias.

Seguem transcrições de evidências relacionadas a esse tópico:

- **EP_04** – *"[...] the use of ASCATs in CI produces a "stronger" message in case the tool reveal warnings, letting everybody be aware about the issue and, in some cases, producing build breakages. At the same time, however, such an effect suggests an use of ASCATs with parsimony to avoid many unnecessary build breakages."*

- **EP_06** – *"Respondents indicate (Q2.5) that code reviews are the premier way for detecting code smells (92%), while 63% of the respondents also use static analysis tools. "*

MP11. Coleta de méticas de qualidade

A qualidade é interpretada e entendida de maneira diferente por cada um. Os desenvolvedores coletam métricas para o monitorar a qualidade do código produzido, algumas dessas métricas são a quantidade de código duplicado, que tradicionalmente também é considerada uma espécie de cheiro ruim, a complexidade ciclomática, número de parâmetros de função e *Lines of Code* (LOC).

Seguem transcrições de evidências relacionadas a esse tópico:

- **EP_06** – *"[...] developers collect a series of metrics to monitor source code quality (Q2.8). The main metrics used are reported in Fig. 8. Surprisingly, the most important metric is the amount of duplicated code (78%) which traditionally is considered as a kind of bad smell too. Other than that, the cyclomatic complexity (69%, again, indicator of some code smells such as Complex Method) and number of function parameters (51%,indicator of Long Parameter List bad smell). Only 44% of respondents mention LOC"*

MP12. Testes automatizados

O teste automatizado é um fator importante que afeta a relação custo-benefício da integração contínua, pois os testes estão intimamente relacionada a ela, logo, CI sem tais testes não deve ser considerado IC de forma alguma (EP_27). Zhao *et.al* (EP_27) diz que a CI é frequentemente introduzida juntamente com a automação de testes.

A adoção de testes automaticos contribui como um impacto positivo na qualidade geral do código de produção, permitindo implantações mais rápidas e com alta qualidade (EP_28, EP_46, EP_49). Itkonen et al. (EP_49) identificaram o teste automático como um dos fatores causam a diminuição do risco de liberação e implantação dos aplicativos, pois os testes reduziram o risco de descobrir bugs após as implantações de produção.

Seguem transcrições de evidências relacionadas a esse tópico:

- **EP_27** – *"Automated testing is an important factor that affects the cost-effectiveness of CI and much effort has been devoted to improve the quality and efficiency of automated testing in CI."*

- **EP_28** – *"Automation in testing and deployment contributed to the positive impact on the overall quality of the production code."*

MP13. Definir Estratégias de testes

As estratégias de teste servem para guiar a equipe de desenvolvimento a um objetivo em comum: eliminar o máximo possível de bugs e desvios de implementação. Os desenvolvedores usam estratégias de teste específicas raramente ou não usam nenhuma estratégia, porém dependendo do recurso em teste, eles escolhem qualquer estratégia é a mais adequada. Quanto ao nível de cobertura os desenvolvedores que tentam atingir pelo menos 80%.

Seguem transcrições de evidências relacionadas a esse tópico:

- **EP_06** – *"We found that developers make use of specific testing strategies such as black box testing relatively seldom (Q3.5). 52% of the respondents say they do not use any strategy.[...] Most of the respondents picked multiple options [criteria white box testing] indicating that depending on the feature under test, they choose whichever strategy is most suitable.[...] Overall, about statement coverage (Q3.12), 84% of the respondents indicated they try to achieve a coverage level of at least 80%."*

MP14. Melhorar a atividade de teste

Essa seção discute 17 práticas propostas na literatura para melhorar a fase de teste durante a CI. Vassallo et al., Amrit e Meijberg (EP_06, EP_08) sugerem que o uso da prática de desenvolvimento ***Test Driven Development*** **(TDD)** indicam um aumento na qualidade do código juntamente com um efeito inconclusivo na produtividade do desenvolvedor, pois facilita a localização de mais defeitos (*Defect Reduction*) e também na redução do tempo necessário para corrigir os defeitos (*Defect lead and throughput*).

Seguem transcrições de evidências relacionadas a esse tópico:

- **EP_06** – *"Generally, our survey answers suggest that quality assurance through testing is a crucial concern at ING NL.[...] The majority of developers mention they use TDD, although we do not know whether they are strictly applying TDD. At the same time, quality assurance in the form of (manual) testing requires a significant portion of the allocated time for a sprint."*

- **EP_08** – *"The outcomes of the empirical studies of TDD implementation seem to indicate an increase in code quality along with an inconclusive effect on developer productivity[...] Though the inferential statistics are not conclusively in favor of the TDD and CI case (CS2), the descriptive statistics point to an overall improvement in not only finding more defects (Defect Reduction), but also in shortening the time required to fix the defects (Defect lead and throughput)."*

Vassallo et al. (EP_06) explica que os desenvolvedores confundem o conceitos de ***Behavior Driven Development* (BDD)** e TDD, no entanto, o BDD é uma técnica de desenvolvimento de software, onde os programadores desenvolvem o software direcionados por comportamentos, deferente do Chen (EP_42) que sugerem que as duas técnicas de desenvolvimento sejam executadas em conjunto.

Seguem transcrições de evidências relacionadas a esse tópico:

- **EP_06** – *"Casual evidence from another context (not at ING NL) suggests that, some developers were referring to acceptance testing with the Framework for Integrated Testing (FIT) [36] as TDD, but meant Behavior-Driven Development (BDD) [37]."*
- **EP_42** – *"For example, we ask new teams to use Kanban, Test Driven Development (TDD) (Beck, 2002), Behavior Driven Development (BDD) (Wynne and Hellesoy, 2012), and other, similar development methodologies."*

No estudo, Chen (EP_42) defende a **inclusão das verificações de segurança automatizadas** em um pipeline CD, sendo executada nos estágios iniciais, produzindo uma trilha de auditoria das alterações feitas para cada confirmação de código, incluindo quando a varredura de segurança foi conduzida, o que foi verificado e quais foram os resultados da verificação. Shahin et al. (EP_14) complementam informando que tal prática alivia a preocuxapação dos lideres de negócio quanto nível de qualidade do produto de software, tornando o pipeline CD atraente para eles.

Seguem transcrições de evidências relacionadas a esse tópico:

- **EP_14** – *"Whilst the main business leaders' concern is around quality level, our results suggest that integrating automated quality checks and security test in both development and operations processes can alleviate this concern and to a large extent make continuous deployment compelling to business leaders."*

- **EP_42** – *"With CD, the execution of security checks is automated. The automation is built into one of the early stages of the CD pipeline. Thus, every code commit goes through the security check."*

Os **testes unitários** são testes de caixa branca que verificam a lógica das unidades alvo, sendo executados para cada commit do código (EP_20, EP_28, EP_39, EP_42). As unidades podem ser métodos, classes, funcionalidades, módulos e etc. Mårtensson, Ståhl e Bosch (EP_30) afirmam que os testes desse tipo são vistos como "ferramentas de desenvolvedor" e devem ser executados pelos desenvolvedores antes de commitar o código e usados como teste de regresão. Shahin, Babar e Zhu (EP_21) afirmam que se os testes unitários forem mal elaborados e mal escritos, esses se tornar testes de longa execução ocasionando um aumento no tempo de ciclo do pipeline de implantação, bem como a diminuição do ciclo de feedback.

Seguem transcrições de evidências relacionadas a esse tópico:

- **EP_21** – *"Long running tests can increase the cycle time (i.e., the time needed to transfer code from code repository to production) of continuous deployment pipeline as well as slow down the feedback cycle in deployment pipeline. That is mainly because tests are poorly designed and written. For example, a participant reported that they were able to address the slow cycle of feedback from continuous deployment pipeline by applying two approaches: (i) creating much smaller unit tests, in which gave them immediately feedback;[...]"*

- **EP_42** – *"Unit tests are executed for every code commit. Acceptance tests are executed for each code commit that passes the unit tests."*

O **teste de integração** é um tipo de teste de caixa preta que sucede o teste unitário e têm por objetivo encontrar falhas de integração entre as unidades. Portanto, o principal trabalho realizado pelas ferramentas de CI é validando do código produzido através da execução dos testes unidade e de integração (EP_28, EP_36).

Seguem transcrições de evidências relacionadas a esse tópico:

- **EP_28** – *"A CI server validates the code with unit and integration tests and stores a Debian package to an internal repository."*

- **EP_36** – *"For CI builds, the main work done by the CI tools is integration testing, so we parsed the CI build scripts8 to distinguish between deployment builds (the aim of this CI build is to deploy images) and general test builds."*

Teste de cobertura é uma métrica em testes de software capaz de medir a quantidade de testes realizados por um conjunto de testes em toda a base de código. Humble e Farley (2014) definem que a cobertura de testes deve ser de no mínimo 75% de toda a base de código, afim que de os desenvolvedores tenham um bom grau de confiança do funcionamento da aplicação, já para Chen (EP_17) o ideal é, no mínimo, 90% de cobertura, caso contrário, o pipeline do implantação falhará na compilação.

Quando inserida no pipeline de implantação a equipe pode estipular um valor mínimo de cobertura de testes que o código deve ter (EP_32), caso a cobertura seja menor que a estipulada a o pipeline do CD falhará na construção (EP_17).

Seguem transcrições de evidências relacionadas a esse tópico:

- **EP_17** – *"Test code is reviewed as rigorously as application code. For any code change, if the test coverage (both line and condition) is below 90%, the CD pipeline will fail the build."*

- **EP_32** – *"BigCorp had put plenty of resources to verify the quality of their product before giving it to the customers. First, the organization had dedicated testers that created automated test cases but also performed manual testing on the product. Second, the organization had a definition of done that included test coverage criteria for automated tests."*

Os ***smoke test*** ou teste de fumaça mencionado nos estudos (EP_15, EP_32) são um subconjunto de testes que são executados rapidamente com frequência, em vez de executar suítes de teste mais extensas, que levariam mais tempo para serem executadas. Os teste de fumaça cobrem as funcionalidades mais importantes, sendo usado para avaliar se tais funcionalidades parecem estar funcionando corretamente (EP_32). Callanan e Spillane (EP_15) dizem que esses testes podem ser usados para monitorar ferramentas a serem executadas ou para o pessoal de operações usar durante a manutenção ou solucionar problemas.

Seguem transcrições de evidências relacionadas a esse tópico:

- **EP_15** – *"The script was a simple Linux shell script named smoketest.sh that assumed the service it was testing was already running on localhost, indicating failure via a*

nonzero exit code if errors occurred.[...] Smoke tests were to be "nondestructive"— that is, using read-only access or writing only to previously approved test data. These tests were also available for monitoring tools to execute or for operations staff to use during maintenance or firefighting."

- **EP_32** – *"[...] that early in the development, SmallOrg did not focus on unit testing at all and they had increased the coverage only afterwards. Instead of aiming at a high test coverage, SmallOrg had created smoke tests which allowed fast verification that nothing was critically broken."*

Build test ou testes de compilação é um conjunto de testes que são executados para determinar se o código foi compilado corretamente (EP_20). Além de serem executados diversos tipos de teste como testes de unidade, testes de integração, *smoke test*, o teste de compilação pode ser executado para garantir que a construção com as novas alterações funcione corretamente. Esses testes podem ser usados quando uma compilação completa consome muito tempo e recursos apenas para testar as dependências com alguns níveis de profundidade (EP_20).

Seguem transcrições de evidências relacionadas a esse tópico:

- **EP_20** – *"As one might expect, FB applies numerous types of tests, including unit tests, static analysis tests, integration tests, screen layout tests, performance tests, build tests, as well as manual tests.[...] Moreover, a test is run to ensure a build with the changes works correctly. However, since a full build is time and resource intensive, the build test here only tests dependencies a few levels deep."*

O **teste de regressão** é uma técnica de teste de software que consiste na execução de algum subconjunto de testes para garantir que a adição de uma nova funcionalidade não gere defeitos em funcionalidades já testadas e que não sofreram modificações. Em um pipeline de implantação os testes de regressão passam a ser o conjunto de testes automatizado que executado toda vez que um código é commitado garantindo que não propagou nenhum defeito ao código pré-existente.

Seguem transcrições de evidências relacionadas a esse tópico:

- **EP_20** – *"Integration tests: These standard (blackbox) regression tests test key features and key flows of the app. They typically run on simulators and are not necessarily device specific."*

- **EP_26** – *"Nightly regression testing of core functionality kept pace with development and supported both functional testing and system-to-system integration. As defects were found in end-to-end business scenarios, responsive resolutions were managed in hours or days, not the weeks typical for larger enterprise systems."*

Testes de sistema são testes caixa preta realizados em um sistema integrado completo para avaliar se está em conformidade com os requisitos. No pipeline de implantação esse tipo de teste é executado depois dos testes de unidade e de integração, seguindo essa ordem (EP_30). Mårtensson, Ståhl e Bosch (EP_30, apud Rathod e Surve, 2015) disseram que só por meio dos testes de sistema que a qualidade e segurança de um produto de software pode ser garantida.

Seguem transcrições de evidências relacionadas a esse tópico:

- **EP_30** – *"Rathod and Surve present a different approach: "It is only through efficient system testing that the quality and safety of a software product can be guaranteed."[...] Unit/component tests followed by system tests to check the developers' software changes. System tests of vital functions (eg, start up the system or drive around) secure stability and integrity in the system, which is monitored by, eg, a test manager or QA department."*

Os **testes de aceitação** tem como objetivo validar se as funcionalidades do produto estão de acordo com os requisitos e processos de negócios do usuário para determinar se o sistema satisfaz ou não os critérios de aceitação estabelecidos (EP_15, EP_42). Esses testes devem ser executados em um ambiente similar ao de produção, no entanto, dependendo das configurações específicas do ambiente de produção, isso pode dificultar a construção dos ambientes de testes (EP_16, EP_21).

Além disso, Lwakatare et al. (EP_16) dizem que a falta cobertura de teste de aceitação totalmente automatizada, exigirá que muito tempo seja gasto em teste de aceitação manual.

Seguem transcrições de evidências relacionadas a esse tópico:

- **EP_15** – *"The compliance tests would run on every commit, during deployment to the acceptance test environments, blocking that release candidate if it didn't meet one or more of the standards criteria."*

- **EP_42** – *"While unit tests and acceptance tests have been extensively discussed and widely practiced in CD, testing non-functional requirements has received considerably*

less attention. However, unit tests and acceptance tests are mainly intended to ensure functional requirements.[...] Acceptance tests are executed for each code commit that passes the unit tests."

Teste funcional é uma técnica de teste de software do tipo caixa preta usada para validar se o sistema foi construido conforme os requisitos especificados. Diferentemente do teste de aceitação que é considerado uma atividade de validação, pois garante ao cliente que o produto construido atende as suas necessidades, o teste funcional é uma atividade de verificação e servindo como garantia à equipe de desenvolvimento de que a aplicação funciona corretamente.

Seguem transcrições de evidências relacionadas a esse tópico:

- **EP_29** – *"One of the interviewees (P12) described how improving the quality of tests could help to accelerate the feedback cycle in the CD pipeline. He highlighted two practices: (i) designing smaller test units that result in immediate feedback, (ii) decreasing the number of functional tests."*

Callanan e Spillane (EP_15) mencionam o **teste de recuperação de desastre** como sendo um tipo de teste não funcional. Esses testes verificam a capacidade do software se recuperar de falhas de software, hardware, falhas de rede, entre outras.

Seguem transcrições de evidências relacionadas a esse tópico:

- **EP_15** – *"Other forms of nonfunctional testing, such as security testing and disaster recovery testing,[...]"*

O ***snapshot test*** é usado para garantir que a interface do usuário não mudou. Esse tipo de teste gera imagens de visualizações de tela e componentes (*snapshot*), que são comparados a um arquivo de *snapshot* de referência armazenado ao lado do teste, pixel por pixel, falhando se os dois *snapshots* não coincidirem (EP_20).

Seguem transcrições de evidências relacionadas a esse tópico:

- **EP_20** – *"Snapshot tests: These tests generate images of screen views and components, which are then compared, pixel by pixel, to previous snapshot versions [14, 15]."*

O **teste de performance** é um tipo de teste que é realizado para verificar o tempo de resposta de uma aplicação, identificar gargalos e determinar a escalabilidade e confiança da aplicação levando em consideração uma carga (EP_50). Rossi et al. (EP_20) declarou o teste

de performance como um dos mais importantes, no entanto, Chen (EP_42) o pipeline de implantação vem sendo adotado sem os testes de aceitação e de performance.

Diferentemente de teste que avaliam os requisitos funcionais de um sistema, o teste de performance avalia requisitos não funcionais, ou seja, a definição dos critérios de aprovação ou reprovação desse teste se torna dificil. Os resultados do teste de performance pode variar de um teste para outro devido a variações inevitáveis nos ambientes de teste, como por exemplo as condições da rede.

Seguem transcrições de evidências relacionadas a esse tópico:

- **EP_42** – *"Every time the developers see the pipeline view, the gaps in their CD adoption are clear. For example, they can easily see that the acceptance and performance stages are empty. We have often seen developers start discussions about how they could fill the gaps by writing automated acceptance tests and performance tests."*

- **EP_50** – *"After successful completion of system testing, performance tests are run to catch performance issues as early as possible. The performance tests are executed by the developers in non-virtual environments for reproducibility. Automated measurements are made and compared with historical data."*

Os **testes de capacidade** são criados para testar se a aplicação e o ambiente podem lidar com o número de usuários e transações a quais foram projetados para lidar (EP_20). Esse tipo de teste pode ajudar a determinar quando o ambiente precisa ser reforçado para atender à demanda.

Seguem transcrições de evidências relacionadas a esse tópico:

- **EP_20** – *"Capacity tests: These tests verify that the app does not exceed various specified capacity limits"*

O **teste de conformidade** é tipo de teste funcional, caixa-preta, que determina se a aplicação cumpre com os padrões especificado (EP_20). Esses testes verificam se o produto entregue em cada fase do ciclo de desenvolvimento está de acordo com os requisitos acordados.

Seguem transcrições de evidências relacionadas a esse tópico:

- **EP_20** – *"Conformance tests: These tests verify that the app conforms to various requirements."*

Lwakatare et al. (EP_16) diz que com a adoção de práticas contínuas as empresas passam a ter a oportunidade de verificar rapidamente se seus novos recursos de software são úteis para os clientes e adotar práticas como o **teste A/B** para conduzir a experimentação de recursos. O teste A/b é uma prática uma prática em que os usuários são atribuídos aleatoriamente a uma das duas versões: a atual e uma "desafiante", com modificações, com o objetivo de descobrir qual das versões apresenta maior taxa de conversão.

Seguem transcrições de evidências relacionadas a esse tópico:

- **EP_16** – *"[...] paradigm change towards continuous deployment gives companies the opportunity to quickly verify whether their new software features are useful to customers and adopting practices such as A/B testing to conduct feature experimentation [5]. A/B testing is a practice where users are randomly assigned to one of the two variants of the system for experimentation e.g. feature usage experimentation [6].[...] Web companies with DevOps practices not only monitor the performance of infrastructure but also conduct experiments regarding feature usage through A/B testing and canary releases [1], [2]."*

MP15. Paralelização de Testes

A paralelização de testes significa executar testes automatizados em paralelo em vez de serialmente, fazendo com que o tempo de execução dos testes seja reduzido. Essa prática necessitada de recursos suficientes para executar os testes, esses podem ser executados em uma ou várias máquinas (LAUKKANEN; ITKONEN; LASSENIUS, 2017).

Seguem transcrições de evidências relacionadas a esse tópico:

- **EP_14** – *"The participants also mentioned the need to test all types of applications (for example mobile testing as it is fragile and expensive to automate), techniques and tools that enable parallelization of automated testing and infrastructure automation testing."*
- **EP_32** – *"SmallOrg had reduced the build time by building only components that had changed after previous run and by running tests in parallel."*

MP16. Microsserviços

A arquitetura microsserviços tenta quebrar sistemas complexos em serviços pequenos, autônomos e implantáveis de forma independente, sendo o estilo arquitetônico inicial e promissor para práticas de CD. A mudança de uma arquitetura monolítica para para uma arquitetura de microsserviços, observa-se uma maior capacidade de implantação, modificabilidade e resiliência à erosão da arquitetura.

Com a adoção de microsserviços surgem dificuldades no monitoramento e na determinação de métricas úteis, colocando muita sobrecarga de operações, se fazendo necessário ter uma equipe de operações madura para adotá-lo (EP_28). Portanto, não se faz necessário adotar microsserviços ou dividir de software monolíticos em microsserviços para se adotar práticas de CD(EP_29).

Seguem transcrições de evidências relacionadas a esse tópico:

- **EP_17** – *"[...] we moved from a monolithic architecture to a Microservices architecture [8]. Rather than building large monolithic applications, we organize systems into small, self-contained, singleresponsibility units that can be independently developed, tested, and deployed.[...]After we moved our applications to a Microservices architecture, we observed increased deployability, modifiability, and resilience to architecture erosion[...]"*

- **EP_21** – *" It has been said that microservices architectural style is the first architectural style for CD practice [13].[...]since microservices style puts a lot of operations overhead in terms of like monitoring these services, and administrating them, it is needed to have mature operations team to adopt it."*

MP17. Build automatizado

O *build* ou compilação de um projeto é uma atividade composta por vários passso, que inclui a compilação do código, execução de testes, empacotamento, geração de artefatos de documentação, entre outros. Essa atividade deve ser repetida a cada mudança realizada na aplicação.

Fazer com que os códigos fontes se transformem em um sistema em execução pode muitas vezes ser um processo complicado se executado manualmente, pois apresentam risco de falha devido a erro humano, portanto, tal tarefa pode ser automatizada (EP_48; FOWLER, 2006). A

automatização do processo de *build* é uma das principais práticas que compõe o CI (FOWLER, 2006), e deve ser executado automaticamente a cada commit, reduzindo o risco de compilações defeituosas devido à configuração manual incorreta e o impacto das alterações do ambiente nas versões existentes (EP_39, EP_48).

Seguem transcrições de evidências relacionadas a esse tópico:

- **EP_28** – *"New merges to the master branch trigger the CI server to automatically build and subsequently deploy to the test environment, where unit tests and end-to-end tests against test database are executed."*
- **EP_39** – *"Build process is run automatically on commit.[...] The commit triggers the build process in GitLab that runs the code analysis and unit tests (#6, 20)."*

MP18. Implantação automatizada

A prática de autimatizar a implantação é o que permite implantar o software em um ambiente de testes ou de produção com apenas um clique, sendo essencial para reduzir o risco de implantações em produção e fornecer feedback rápido sobre a qualidade do software, permitindo que as equipes realizem testes abrangentes o mais rápido possível após as alterações. Com a adoção dessa prática o processo torna-se repetível e confiável (EP_28), pois o computador passa então a executar as tarefas de implantação sem intervenção, diminuindo os risco de falha devido a erro humano.

O mecanismo de implantação automatizado pode ser incorporado ao servidor de integração contínua (CI), além de poder usar diferentes estratégias de implantação (EP_28), como implantação azul-verde e a *rolling upgrade*, ou implantação canária, entre outras.

Seguem transcrições de evidências relacionadas a esse tópico:

- **EP_28** – *"The automated deployment mechanism can use different deployment strategies, but when created, it ensures the reliability and repeatability of system deployment [13,38].[...] Deployment scripts facilitated the automatic installation of software to a target environment by software development teams. Deployments scripts were mostly triggered from the CI system. Software developers in all cases had access and could modify the scripts, which were also version-controlled."*

- **EP_48** – *"Also we need to automate deployment process which covers creating and running service containers in test environment, so developer and operator can test software without any extra effort.[...] To automate service deployment in test environment, test environment should exposes Docker remote API."*

MP19. Definir uma Estratégia de Implantação

No lançamento de uma versão de software ao consumidor, introduz-se o risco de vulnerabilidades, problemas, bugs e software sem desempenho acontecerem em produção, e portanto, havendo vários motivos para reverter uma implantação ou produzir um *hotfix*.

Definir uma estratégia de liberação que funcione reduz o medo e o risco de mudanças no lançamento de novos recursos para um cliente. Os autores do (EP_13) sugeriram diversas estratégias de implantação, as quais serão descritas a seguir, dentre elas está a ***"Feature Flags"*** que trata-se de uma técnica que permite que novos recurso sejam implantados em um ambiente de produção, restringindo sua disponibilidade através de um sinalizador, possibilitando habilitar ou desabilitar o novo recurso para usuários específicos sem reiniciar o software ou implantar um novo código.

- **EP_13** – *"Microsoft often deploys large architectural changes, using a combination of dark launches and feature flags. With a feature flag, a feature is deployed but disabled until it's ready for release; the developer turns the feature off and on through a configuration server. This practice lets Microsoft avoid dealing with integration issues or maintaining long-running feature branches"*

Os autores (EP_13, EP_28) sugerem a estratégia "**implantação canário**" que reduz o risco da introdução de uma nova versão do software em produção, fazendo o lançamento gradual da mesma para uma pequena parte do conjunto de usuários antes de implantá-la em toda a infraestrutura e torná-la acessível a todos.

Seguem transcrições de evidências relacionadas a esse tópico:

- **EP_13** – *"A practice in which code under test is first released to a small batch of real users. If the metrics deviate from nominal ranges, routing to canary release might automatically halt."*

- **EP_28** – *'Canary deployment used in Case E ensured testing in production environment with the users.[...] such as canary release that exposes software changes incrementally to a portion of users before deploying them to entire user base."*

A estratégia de implantação conhecida como ***"dark launch"*** sugeridas nos (EP_13, EP_50), descreve o processo de liberar uma funcionalidade ou recurso para um subconjunto de usuários afim de obter respostas do usuário, sendo que esses usuários não sabem que estão sendo testados e não têm a nova funcionalidade apontada para eles de forma alguma, por exemplo o lançamento experimental de uma compra de um clique para meus usuários afim de avaliar se irá ocorrer o aumento das vendas.

Dark Launches e Implantações Canários são bastante semelhantes, pois ambos lidam com o lançamento de novas funcionalidades no ambiente de produção para um subconjunto de usuários reais antes de liberar para todos, desvinculando a implantação do lançamento. No entanto, a Implantação Canária são mais comumente usadas para testar novos recursos no back-end, afim de fazer a transição lentamente para uma nova infraestrutura.

Seguem transcrições de evidências relacionadas a esse tópico:

- **EP_13** – *"This dark launch lets the engineer slowly deploy and stabilize small chunks directly during production without impacting the user experience. After stabilization, the engineer can turn on the feature and release it.[...] Instagram often uses dark launches to deploy and stabilize features for up to six months before officially releasing them. Microsoft often deploys large architectural changes, using a combination of dark launches and feature flags."*

- **EP_50** – *"dark launches: A deployment strategy where changes are released during off peak hours; or where code is installed on all servers, but configured so that users do not see their effects because their user interface components are switched off. Such launches can be used to test scalability and performance [14] and can be used to break a larger release into smaller ones."*

Os autores em (EP_28, EP_50) sugeriram outra estratégia de implantação chamada ***"Blue-Green Deployments"*** que é um modelo de implantação que transfere gradualmente o tráfego de usuários de uma versão anterior (azul) da aplicação ou microsserviço para uma nova versão (verde), estando ambas as versões em execução em ambiente de produção idênticos, eliminando o tempo de inatividade devido à implantação do aplicativo. Quando o ambiente

verde estiver com todo o trafego, o ambiente azul ficará offline e poderá servir de standby como opção de recuperação de desastres ou tornar-se o ambiente da próxima atualização.

Seguem transcrições de evidências relacionadas a esse tópico:

- **EP_28** – *"The cases employed different deployment strategies, such as blue-green deployment (Cases C and E)[...]"*
- **EP_50** – *"blue-green deployments: A deployment strategy where a defective change to a production environment (blue) can be quickly switched to the latest stable production build (green) [15]."*

Diferente da *"Blue-Green Deployments"* que cria um ambiente separado para a uma nova versão, sem afetar a antiga, onde são realizados testes na nova versão e, uma vez pronta, inicia-se o encaminhamento dos usuários para a nova versão. Já o ***"Rolling Upgrade"*** apresentado no (EP_28), possibilita que a nova versão seja implantada gradualmente nos servidores de produção fazendo com que o *cluster* torne-se heterogêneos, ou seja, havendo servidores com a versão antiga e outros com a nova versão, até a nova versão ter sido implantada todos os servidores. Essa estratégia tem como objetivo final de substituir lentamente todas as versões antigas pelas novas.

Seguem transcrições de evidências relacionadas a esse tópico:

- **EP_28** – *"The cases employed different deployment strategies, such as blue-green deployment (Cases C and E), rolling upgrade (Case B)[...]"*

A estratégia de implantação ***"Staging/Baking"*** mencionada no (EP_50) consiste de se ter uma etapa antes do lançamento para a produção, como uma especie de ensaio final. Essa etapa é denomidada *staging* ou "encenação" a qual a nova versão do software é testada tendo com o objetivo de garantir que a aplicação atenda aos requisitos e expectativas dos negócios, em um ambiente identico ao de produção (isso significa que as configurações da máquina entre a encenação e a produção devem coincidir).

Seguem transcrições de evidências relacionadas a esse tópico:

- **EP_50** – *"staging/baking: A stage in the deployment pipeline where a new version of software is tested in conditions similar to a production environment. An example of this is called shadow testing where production traffic is cloned and sent to a set of shadow machines that execute newer code than production. Results between production*

and shadow environments can be automatically compared and discrepancies reported as failures."

MP20. Definir Mecanismo de Rollback

Quando uma nova funcionalidade gera defeitos na produção deve-se ter um plano para reverter ao estado anterior. A adoção de uma estratégia de reversão deve ser definida em conjunto com a de implantação, pois a essa estratégia tende a espelhar a de implantação. Como mencionado anteriormente algumas estratégias de implantação, como a implantação canário, *dark launch*, implantação *blue-green* e a *rolling upgrade*, por exemplo, possibilitam e reversão da implantação por definição.

Seguem transcrições de evidências relacionadas a esse tópico:

- **EP_** – *"The cases employed different deployment strategies, such as blue-green deployment (Cases C and E), rolling upgrade (Case B) and canary deployment (Case E). Roll-back mechanism was said to be in place in Cases B, C, D and E."*

5.2.3 Q3. Quais ferramentas foram empregadas para projetar e implementar pipelines de implantação?

Esta seção apresenta as conclusões para responder a Q3. O sucesso da adoção de práticas contínuas nas empresas depende muito de pipelines de implantação (SHAHIN; ALI BABAR; ZHU, 2017). Portanto, a escolha adequada de ferramentas para construir esse pipeline pode ajudar a mitigar alguns dos desafios da adoção e implementação de práticas de integração, entrega e implantação contínuas. Investigamos as ferramentas de implantação e as ferramentas que são usadas no pipeline de implantação que foram relatadas na literatura. Para facilitar o entendimento, adotamos o termo pipeline de implantação, para descrever o processo de transferir o código do repositório de código para o ambiente de produção.

Embora a automação seja uma prática crítica em um pipeline, às vezes tarefas manuais (por exemplo, tarefas de garantia de qualidade) são necessárias. É importante lembrar que não existe um pipeline único ou padrão, visto que este pode variar de projeto para projeto e de empresa para empresa. Por isso, a equipe deve escolher a melhor ferramenta para a sua situação. Deve-se notar que as ferramentas relatadas nesta seção são, em sua maioria, fontes

abertas e ferramentas comerciais existentes, que visam formar e implementar um pipeline de implantação.

Conforme Apêndice E, foram encontradas 63 ferramentas de apoio para adoção de pipelines de implantação as quais foram categorizadas em 9 categorias: (i) gerenciamento de projetos; (ii) Sistema de Controle de Versão, em inglês *Version Control Systems* (VCS); (iii) ferramenta de gerenciamento e análise de código; (iv) construção do software (*build*); (v) servidor de integração contínua; (vi) ferramenta de teste; (vii) repositório de artefatos; (viii) configuração e provisionamento; e (ix) monitoramento.

A primeira categoria a ser discutida é o gerenciamento de projetos. As ferramentas desta categoria permitem o monitoramento de tarefas e acompanhamento de projetos garantindo o gerenciamento de todas as suas atividades em único lugar entre outras funções. Para ajudar no gerenciamento do projeto de software foram encontradas 3 ferramentas: Jira, Atlassian Confluence e IBM Rational Team Concert (RTC) relatadas em 3 artigos, sendo que cada ferramenta foi citada em um artigo.

Definido o que deverá ser desenvolvido, os desenvolvedores iniciam o trabalho de codificação enviando continuamente seus códigos para o repositório de código. O sistema de controle de versão Git[2] é usado em 22 artigos, no entanto, 7 desses estudos não especificaram qual ferramenta estavam usando, relatando apenas que usando o Git. Os gerenciadores de repositório de software os mais populares foram GitHub, GitLab e BitBucket com 10, 5 e 2 citações respectivamente.

Para a automação da tarefa de construção (*build*) foram encontradas somente 3 ferramentas das quais destaca-se o Maven com 5 citações. As ferramentas relatadas nesta categoria não podem ser usados em todos os projetos, visto que são voltadas para a linguagem Java.

Encontramos 7 artigos (EP_05, EP_06, EP_10, EP_12, EP_24, EP_28 e EP_31) que usaram ferramentas de análise como parte do pipeline de implantação para compor o processo de construção (build) do software. O estudo (EP_05) relatou que integrou o SonarQube ao servidor de CI Jenkins para realizar a análise estática automática.

Por meio de servidores CI, é possível acionar automaticamente o processo de construção e executar testes de unidade quando uma mudança for realizada no repositório de código. O Jenkins ganhou mais atenção entre servidores de CI existentes na literatura, sendo relatado em 17 artigos, seguido pelo Travis CI, CircleCI com e GitLab CI/CD e outros servidores de CI foram relatados em um estudo cada.

[2] Git, acesso em: <https://git-scm.com/>

Quanto as ferramentas associadas a execução dos testes, foram relatadas 9 ferramentas cada uma com uma citação. Ressalta-se que em todas as citações de ferramentas estavam integradas ao pipeline de implantação. Observa-se que nos estudos, há diversos tipos de testes automatizados compondo o pipeline além do testes unitários e de integração, como por exemplo, testes de comportamento com o Cucumber (EP_28) e teste de desempenho utilizando o Apache JMeter (EP_10).

Com a execução do pipeline o software em desenvolvimento será construído e testado diversas vezes em diversos aspectos, resultando ao final da execução do pipeline um arquivo que pode ser disponibilizado em produção. Esse arquivo ou artefato deve ser armazenado em um repositório de artefatos JFrog Artifactory (EP_06 e EP_24) o que permite rastrear os artefatos do desenvolvimento à produção, facilitando a armazenagem e recuperação.

Com a construção e execução dos testes automatizados, a automatização da tarefa de deploy e configuração dos ambientes se faz necessária, como um dos preceitos da entrega contínua. A equipe de desenvolvimento deve achar natural tanto a integração dos códigos quanto a implantação (HUMBLE; FARLEY, 2014). Nesta categoria a ferramenta que mais se destacou foi o Docker, seguido pelo Puppet a diferença de apenas uma citação. Ansible, AWS Cloud, Vagrant tiveram 3 citações seguido pelo Chef com 2 citações, as demais ferramentas obtiveram apenas uma citação cada.

Por fim, após o sistema ser disponibilizado em produção, a atividade de monitoramento do bom funcionamento se faz necessária. Realizar tal tarefa sem uma boa ferramenta de apoio se torna cansativo, devido a isto acrescentamos a busca por ferramentas dessa categoria. Embora tenham sido relatadas 11 ferramentas distintas, o número de citações nos artigos foi relativamente baixo. As ferramentas que se destacaram foi o New Relic e o Nagios com duas citações cada, as demais ferramentas obtiveram apenas uma citação cada.

5.3 AMEAÇAS À VALIDADE

Esta revisão sistemática da literatura sofre ameaças à validade. Em primeiro lugar, a seleção dos artigos foi feita estritamente usando uma *string* de busca específica com palavras-chave para práticas contínuas e filtros, executada em bases de dados online. O resultado foi uma lista gerenciável de artigos para analisar, no entanto, teria sido apropriado utilizar em conjunto uma técnica como "*snowballing*" para analisar os artigos e buscar suas referências e os artigos que os referenciam. A lista de estudos teria sido mais abrangente e mais relacionada à

pesquisa. De fato, alguns estudos encontrados pela *string* não foram realmente úteis, mesmo que correspondessem à consulta.

A etapa de seleção dos estudos pode ser influenciada pelo julgamento subjetivo dos pesquisadores sobre se um artigo atende ou não aos critérios de seleção para inclusão ou exclusão (SHAHIN; ALI BABAR; ZHU, 2017). Os possíveis vieses na seleção dos estudos foram abordados seguindo estritamente o protocolo de revisão predefinido, registrando os motivos de inclusão e exclusão, embora a revisão tenha sido executada apenas por um pesquisador.

A execução de uma RSL por apenas um pesquisador representa uma limitação, mas é previsto por Kitchenham (2007) para alunos de PhD que utilizam revisão sistemática (nada foi dito sobre alunos de mestrado). Segundo a autora, basta que o orientador da tese, assim como outros envolvidos, participe da revisão do protocolo e revise partes da execução da revisão. Neste sentido, o orientador revisou todo o protocolo de estudo deste trabalho, bem como a síntese dos dados e o relatório de resultados;

O viés dos pesquisadores na extração de dados pode ser uma ameaça básica em qualquer RSL, o que pode afetar negativamente os resultados das RSLs. Para lidar com essa ameaça, criamos um formulário de coleta de dados para extrair e analisar consistentemente os dados para responder às questões de pesquisa desta RSL utilizando a StArt.

Durante a extração dos foram aplicados métodos quantitativos e qualitativos. Deve-se notar que em alguns momentos houve algumas dificuldades na interpretação dos dados extraídos devido à falta de informações suficientes sobre os itens de dados. Tivemos que interpretar e analisar subjetivamente os itens de dados, o que pode ter afetado os resultados da extração de dados. Para reduzir o viés dos pesquisadores na interpretação dos resultados, além de ler o estudo em questão, sempre que possível, também consultamos o site da ferramenta e qualquer vídeo de treinamento (por exemplo Q3) para obter informações mais confiáveis. Deve-se notar que, para outros itens de dados, não tivemos nenhuma interpretação, a menos que os itens de dados tenham sido fornecidos explicitamente pelo estudo (por exemplo, domínio de aplicação).

5.4 DISCUSSÃO SOBRE OS RESULTADOS OBTIDOS

Por se tratar de uma revisão sistemática da literatura que serviu como atualização da junção de 3 outras revisões, foi colocada a uma limitação de tempo para a seleção dos estudos, ficando então a pesquisa limitada a 4 anos de publicações. Embora a execução das *strings* de busca tenham sido realizadas em 2020 não obtivemos estudos desse ano.

Observa-se que o número de artigos retornados na execução das *strings* foi alto, porém houve muitos estudos repetidos e muitos estudos da área de medicina, visto a semelhança de alguns termos com os utilizados na busca.

Quanto a avaliação da qualidade dos estudos, embora tenha sido realizada por apenas um pesquisador com pouca experiência, o protocolo foi revisado por pesquisador mais experiente. A elaboração do formulário de avaliação focou na identificação da metodologia, objetivos, justificativa, contexto da pesquisa, se o estudo estava bem referenciado e se o estudo respondia as questões de pesquisa dessa revisão. O formulário simplificado facilitou a execução da etapa de avaliação da qualidade do estudo, demonstrando qualidade dos estudos no contexto de relevância para esta pesquisa.

A análise do domínio da aplicação demonstrou que as práticas contínuas podem ser aplicadas em qualquer tipo de projeto de software. Observamos também que grande parte dos estudos foram em projetos web, porém vemos sua aplicação em aplicativos móveis, software de voo, financeiro e aplicações embarcadas além de aplicações legadas (EP_05).

Ao serem identificados desafios, práticas e ferramentas, informamos que neste capítulo só foram discutidas as evidencias encontradas nos 54 estudos que extraídos. No entanto, as evidências presentes nas revisões que serviram como base para esta, estão listadas nos apêndices C, D, E.

Com relação aos desafios relatados, observa-se que as categorias Humano e Organizacional e Testes reuniram 50% dos desafios, devido ao foco dado pelas práticas contínuas na atividade de teste e na mudança de cultura que deve ser feita na organização. Embora cada processo de adoção das práticas contínuas seja diferente os desafios relatados servem como um auxilio para que as equipes possam supera-los com mais facilidade ou mesmo evitar que eles ocorram.

Com relação as práticas relatadas, foram relatadas diversas práticas relacionadas ao fluxo de trabalho da equipe, codificação entre outras etapas do desenvolvimento. As práticas que ganharam mais destaque foram as relacionadas com a atividade de testes. Na prática MP14. Melhorar a atividade de testes foram listados 16 tipos de testes e duas metodologias de desenvolvimento. Outras práticas que podemos observar, são a revisão de código, microsserviços e análise estática de código automática que tiveram 5, 7 e 8 citações. Isso indica a importância que os estudos dão para questões relacionadas a testes, revisão de código, análise estática e mudança da arquitetura monolítica para microsserviços.

Embora tenham sido relatadas diversas práticas, em nenhum dos 54 estudos analisados, não foram encontradas descrições das etapas ou do processo realizado para a adoção das práticas

contínuas. Os artigos focavam em realizar estudos sobre o impacto de adotar tais práticas nos projetos de software. A descrição da adoção nos estudos ficava limitada a estrutura do pipeline, as ferramentas e práticas utilizadas, os desafios encontrados e raramente suas soluções.

Ao analisarmos as ferramentas para o apoio a adoção do pipeline de implantação, observa-se algumas características que não estão diretamente relacionadas aos estágios de um pipeline, porém foram adicionadas a fim de fornecer uma arcabouço maior de tipos de ferramentas auxiliam na atividade de desenvolvimento e entrega de um projeto de software. Como por exemplo, a categoria de gerenciamento de projeto auxiliará na rastreabilidade das mudanças; já a categoria de configuração e provisionamento auxiliará na automatização do *deploy* e configuração dos ambiente permitindo assim uma repetibilidade da tarefa de implantação.

Já a categoria de monitoramento, como vimos anteriormente as práticas contínuas podem ser implantadas para sistemas legados (EP_05), e portanto, auxiliam a equipe a terem um *feedback* mais rápido com relação a correção de problemas que possam a vir a ocorrer no ambiente de produção.

Portanto, devido a variedade de ferramentas agrupadas em diversas categorias, ajudará aos profissionais e pesquisadores a adotar práticas contínuas, uma vez que o pipeline a ser construído pode variar de projeto para projeto e de empresa para empresa.

5.5 PROPOSTA DE ABORDAGEM PARA ADOÇÃO DE PIPELINE DE ENTREGA CONTÍNUA

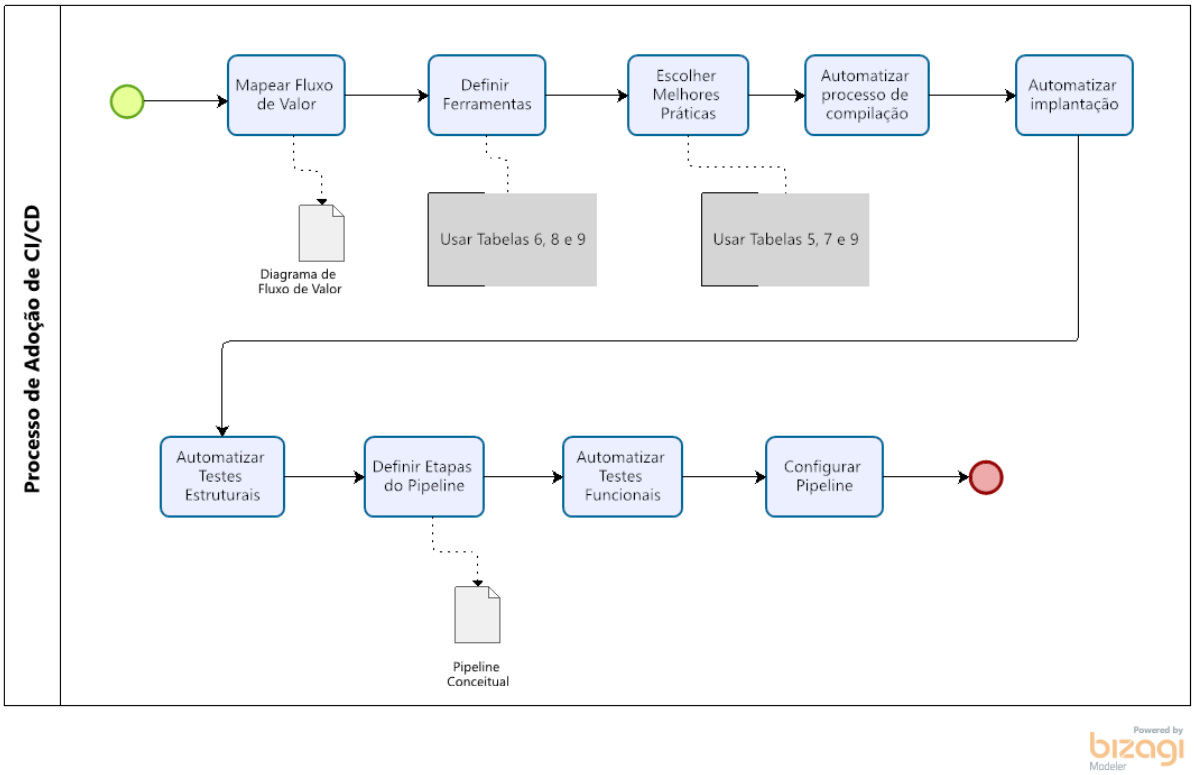
Nessa seção é proposta uma abordagem para orientar a adoção de um pipeline de implantação. A abordagem é estruturada em dois processos:

- **Proposta do Processo de adoção:** usando evidências da experimentação científica e industrial e da literatura, o processo foi construído de forma a auxiliar em quais tarefas devem ser realizadas para adotar o pipeline CI/CD.
- **Abordagem para o Gerenciamento de risco do pipeline:** durante o processo de adoção podem surgir desafios, portanto o gerenciamento do pipeline usa evidências da experimentação científica e industrial, para refinar uma teoria de gerenciamento do pipeline CI/CD que pode ser construída gradualmente e de forma colaborativa.

5.5.1 Proposta do Processo de Adoção

Como nos estudos selecionados não há descrições das etapas ou do processo realizado para a adoção das práticas contínuas. O processo foi construído tendo como base o processo descrito por Duvall, Matyas e Glover (2007) e Humble e Farley (2014). O processo de adoção e suas tarefas são ilustrados na Figura 10, e detalhadas a seguir.

Figura 10 – Processo de Adoção de Pipeline de CI/CD Preliminar



Fonte: Elaborado pelo autor (2022)

Mapear Fluxo de Valor: o primeiro passo é mapear o fluxo de valor do processo de entrega de uma funcionalidade. O mapeamento desse fluxo pode ser completo, ou seja, desde o momento em que ela chega até quando é entregue. Ou parcialmente do *commit* até a entrega em produção. Converse com todos os envolvidos no processo e escreva os passos, incluindo as melhores estimativas sobre tempo decorrido e tempo gasto nas atividades que agregaram valor. Se o projeto for novo o fluxo deverá ser baseado em outro projeto na mesma organização que tem características similares.

Definir Ferramentas: Definir as ferramentas a serem utilizadas na adoção da integração (CI) e entrega contínua (CD). Para isso, deve-se utilizar os componentes da abordagem que

listam as ferramentas (Tabela 13) e as que associam Desafios e Ferramentas (Tabela 4) e Melhores Práticas e Ferramentas (Tabela 5).

Automatizar Processo de Compilação: Com as ferramentas escolhidas automatize o processo de compilação da aplicação de forma que o mesmo possa ser iniciado via linha de comando e por um único comando.

Automatizar Implantação: Com as ferramentas escolhidas automatize o processo de implantação da aplicação de forma que o mesmo possa ser iniciado via linha de comando e por um único comando.

Automatizar Testes Estruturais: Desenvolva testes automatizados de unidade e de integração, para que estes possam proporcionar uma cobertura de no mínimo 75%.

Definir Etapas do Pipeline: Com o fluxo de valor mapeado, defina as etapas do pipeline, ao final, crie um diagrama com as etapas do pipeline.

Automatizar Testes Funcionais: Desenvolva testes automatizados de aceitação, cobrindo caminhos que os testes estruturais não cobriram. Não testar funcionalidades já testadas.

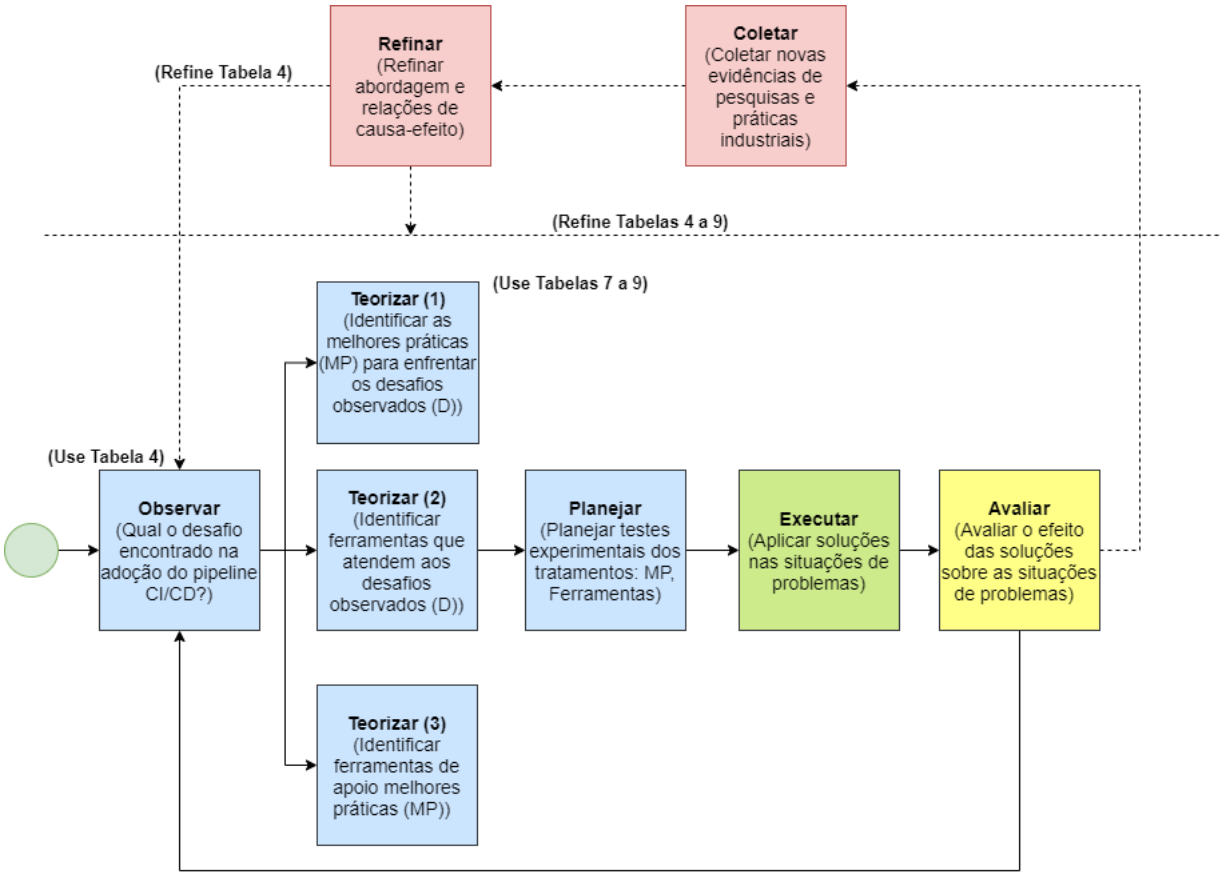
Configurar Pipeline: Por fim, instale e configure o servidor CI, bem como demais ferramentas que sejam necessárias para executar as automações realizadas. As ferramentas devem estar integradas, afim de possibilitar melhores benefícios após adotar as práticas.

5.5.2 Abordagem para Gerenciamento de Risco do Pipeline

Para construir a abordagem de gerenciamento de riscos de adoção do pipeline CI/CD foi usado como base o Método de Análise e Solução de Problemas (MASP) que consiste em uma forma sistemática de realização de ações corretivas e preventivas para eliminar, descobrir e atacar as causas dos problemas, evitando a repetição destes por meio da padronização de procedimentos. Para ajudar na identificação desses desafios, as evidências relacionadas a desafios, melhores práticas e ferramentas (apresentadas na seção 5.2 deste capítulo) foram combinadas nas Tabelas 3, 4 e 5. Essa abordagem permite que a equipe de desenvolvimento, diante da identificação de um desafio, possa verificar as possibilidades que venha a minimizar ou eliminar os obstáculos durante a adoção da integração e entrega contínua. As etapas da abordagem são ilustradas na Figura 11, e detalhadas a seguir.

Etapa 1 - Observar: Observar a situação atual da adoção do pipeline de integração e entrega contínua procurando desafios. Encontrar respostas para a pergunta "Qual o desafio encontrado?". Para isso, devem ser utilizados os Desafios listados na Tabela 11 para orientar

Figura 11 – Abordagem para o Gerenciamento da adoção do pipeline CI/CD



Fonte: Elaborado pelo autor (2022)

a observação. Criar uma lista de desafios relevantes para a situação atual.

Etapa 2 - Teorizar (1, 2, 3): Criar uma potencial solução para cada desafio identificado na fase Observar. Para isso, devem ser utilizados os componentes da abordagem que associam Desafios e Melhores Práticas (Tabela 3), Desafios a Ferramentas (Tabela 4) e Melhores Práticas e Ferramentas (Tabela 5). Neste ponto, as hipóteses são de que a implantação e utilização das Melhores Práticas e Ferramentas selecionadas terão um efeito positivo sobre os problemas identificados. Quantificar (se possível) o efeito desejado de cada solução.

Etapa 3 - Planejar: Construir um plano de aplicação das soluções identificadas na fase Teorizar através de (um conjunto) testes experimentais das hipóteses geradas nas fases anteriores.

Etapa 4 - Execução: Aplicar as soluções sobre a atual situação e coletar dados sobre seus efeitos sobre os problemas.

Etapa 5 - Avaliar: Avaliar a eficácia das soluções, observando seus efeitos sobre a situação do desafio original. Realimentar os resultados para as fases Observar e Coletar. O ciclo de

melhoria pode continuar identificando novos desafios.

Etapa 6 - Coletar: Coletar dados a partir de estudos experimentais, *white papers* e artigos da literatura cinza que relacionem Melhores Práticas e Ferramentas para os Desafios na adoção e utilização de CI/CD. O protocolo de revisão sistemática pode ser utilizado para guiar a coleta de evidências. Os resultados dos experimentos da Fase 5 também contribuem para essa fase.

Etapa 7 - Refinar: Deve-se utilizar as novas evidências a partir de estudos coletados para refinar a abordagem, modificando as Tabela 11 até a Tabela 5, de acordo com os resultados.

Tabela 3 apresenta a relação entre Desafios e Melhores Práticas identificados na literatura. Na primeira coluna são apresentadas as categorias de desafios, e na segunda, as práticas evidenciadas na literatura como possíveis soluções.

Tabela 3 – Desafios e Melhores Práticas

Desafios	Melhores Práticas
D5. Falta de comunicação	MP4. Todos os commits estarem vinculados às tarefas
D7. Falsa sensação de confiança	MP9. Revisão de Código
	MP10. Análise Estática de Código Automática
	MP12. Testes automatizados
D14. Longos tempos de espera para builds	MP5. Commit de código com mais frequência
	MP6. Commits pequenos
	MP7. Trabalhar em Pequenos lotes
D15. Múltiplos Branches	MP2. Padronizar o Fluxo de trabalho no Sistema de Controle de Versões
D16. Falhas de Build	MP5. Commit de código com mais frequência
	MP6. Commits pequenos
	MP7. Trabalhar em Pequenos lotes
	MP17. Build automatizado
D17. Branches longas	MP2. Padronizar o Fluxo de trabalho no Sistema de Controle de Versões
D19. Arquitetura Altamente Acoplada	MP16. Microsserviços
D21. Baixa Cobertura de Teste	MP14. Melhorar a atividade de teste

D22. Falta de Teste de Aceitação Automatizado	MP14. Melhorar a atividade de teste
D23. Testes de Longa Duração	MP13. Definir Estratégias de testes
	MP15. Paralelização de Testes automatizados
D24. Falta de Testes	MP12. Testes automatizados
	MP13. Definir Estratégias de testes
	MP14. Melhorar a atividade de teste
D25. Testes Instáveis	MP13. Definir Estratégias de testes
	MP14. Melhorar a atividade de teste
D26. Falta de Estratégia de Teste	MP13. Definir Estratégias de testes
D27. Teste Inadequado	MP13. Definir Estratégias de testes
	MP14. Melhorar a atividade de teste
D28. Falta de Mecanismo de Reversão	MP20. Mecanismo de Rollback
D29. Processo de Entrega Demorado	MP5. Commit de código com mais frequência
	MP6. Commits pequenos
	MP7. Trabalhar em Pequenos lotes
	MP13. Definir Estratégias de testes
	MP15. Paralelização de Testes automatizados
	MP17. Build automatizado
	MP18. Implantação automatizada
D31. Configuração Manual de Software	MP17. Build automatizado
	MP18. Implantação automatizada
	MP19. Definir uma Estratégia de Implantação
	MP20. Definir Mecanismo de Rollback

Fonte: Elaborado pelo autor, com dados da revisão sistemática (2022)

A Tabela 4 apresenta a relação entre Desafios e Ferramentas. Na primeira coluna são apresentadas as categorias de Desafios, e na segunda, as Ferramentas evidenciadas na literatura para apoiar o gerenciamento.

Tabela 4 – Desafios e Ferramentas

Desafios	Ferramentas
D5. Falta de comunicação	F1. IBM Rational Team Concert (RTC)
	F2. Jira
	F3. Atlassian Confluence
D18. Dependências	F40. JFrog Artifactory
	F41. Nexus Repository Manager
D22. Falta de Teste de Aceitação Automatizado	F32. Selenium
	F33. Cucumber
D26. Falta de Estratégia de Teste	F31. IBM Rational Quality Manager (RQM)
D28. Falta de Mecanismo de Reversão	F42. AWS Cloud
	F43. Puppet
	F44. Vagrant
	F45. Chef
	F46. Docker
	F47. Ansible
	F48. Plataforma System Center Orchestrator
	F49. Chake
	F50. Nolio
	F52. Kubernetes
D31. Configuração Manual de Software	F42. AWS Cloud
	F43. Puppet
	F44. Vagrant
	F45. Chef
	F46. Docker
	F47. Ansible
	F48. Plataforma System Center Orchestrator
	F49. Chake
	F50. Nolio

	F51. SSH
	F52. Kubernetes

Fonte: Elaborado pelo autor, com dados da revisão sistemática (2022)

A Tabela 5 apresenta a relação entre Melhores Práticas e Ferramentas. Na primeira coluna são apresentadas as categorias de Melhores Práticas, e na segunda, as Ferramentas evidenciadas na literatura de apoio às Práticas.

Tabela 5 – Melhores Práticas e Ferramentas

Melhores Práticas	**Ferramentas**
MP4. Todos os commits estarem vinculados às tarefas	F1. IBM Rational Team Concert (RTC)
	F2. Jira
	F3. Atlassian Confluence
	F4. GitHub
	F6. BitBucket
	F7. Deveo
	F8. GitLab
	F9. Apache Subversion (SVN)
	F10. Azure DevOps
MP9. Revisão de Código	F4. GitHub
	F6. BitBucket
	F8. GitLab
	F10. Azure DevOps
MP10. Análise Estática de Código Automática	F11. SonarQube
	F12. CodeSonar
	F13. Gerrit
	F14. Klocwork
	F15. Attack Surface Analyzer
	F16. Microsoft Baseline Security Analyzer

MP11. Coleta de méticas de qualidade	F11. SonarQube
	F12. CodeSonar
	F13. Gerrit
	F14. Klocwork
	F15. Attack Surface Analyzer
	F16. Microsoft Baseline Security Analyzer
MP14. Melhorar a atividade de teste	F31. IBM Rational Quality Manager (RQM)
	F32. Selenium
	F33. Cucumber
	F34. Fortify
	F35. Gatling
	F36. JMeter
	F39. Smokemonster
MP17. Build automatizado	F17. Maven
	F18. Gradle
	F19. Ant
MP18. Implantação automatizada	F41. Plataforma System Center Orchestrator
	F42. AWS Cloud
	F43. Puppet
	F44. Vagrant
	F45. Chef
	F46. Docker
	F47. Ansible
	F49. Chake
	F50. Nolio
	F51. SSH
	F52. Kubernetes

Fonte: Elaborado pelo autor, com dados da revisão sistemática (2022)

5.6 SÍNTESE DO CAPÍTULO

Neste capítulo foram apresentados os resultados da revisão sistemática divididos na análise descritiva e análise de evidências. Na análise descritiva foram apresentados os dados gerais da revisão como: quantidade de trabalhos retornados nas buscas, processo de seleção com o número final de estudos primários, distribuição dos estudos por tipo de publicação ao longo dos anos, locais de publicação e por tipo de estudo, a avaliação da qualidade de aplicação, e por fim, a distribuição do domínio. Já a análise das evidências apresentou e descreveu as evidências identificadas pela revisão sistemática, isto é, apresentou os resultados para cada questão de pesquisa.

As evidências coletadas fornecem à comunidade acadêmica uma melhor compreensão sobre os desafios na adoção de integração (CI) e entrega contínua (CD). E, a partir da análise dos dados coletados nesta pesquisa, possibilitaram a criação de uma abordagem para implantar um pipeline CI/CD, além de apoiar os profissionais e pesquisadores na identificação de desafios relevantes e definição de soluções para os mesmos, utilizando para isso, as melhores práticas e ferramentas que já foram testados por outros estudos primários, em ambientes experimentais e industriais.

A abordagem foi divida em dois processos, o de adoção, elencando as tarefas necessárias para que o pipeline CI/CD seja implantado e o de gerenciamento de risco, que auxilia a equipe de desenvolvimento a gerenciar o riscos relacionados ao processo de adoção baseado no MASP, possibilitando que frente a um desafio, possam verificar as possibilidades de minimizá-lo ou eliminá-lo.

No capítulo seguinte, apresenta-se uma estratégia para adotar integração e entrega contínua, com um estudo sobre o processo de desenvolvimento e do processo de entrega de software, uma estratégia para adotar integração e entrega contínua e uma avaliação do impacto da adoções dessas práticas através do projeto irmão.

6 ESTRATÉGIA PARA ADOTAR INTEGRAÇÃO E ENTREGA CONTÍNUA

O Capítulo 6, apresenta estratégias minímas para adotar um pipeline CI/CD através de um estudo realizado no Instituto Federal do Acre (IFAC) sobre o processo de entrega dos produtos de software, no qual foi adotada a integração e entrega contínua em um projeto piloto, através da abordagem construída no capítulo anterior (ver Capítulo 5) e ao final foi realizada uma avaliação dessas práticas. O capítulo está estruturado conforme as seguintes seções:

1. **Ambiente da Pesquisa: O Instituto Federal do Acre** (seção 6.1): nesta seção é apresentado o IFAC, contando brevemente sua história e a estrutura de Tecnologia da Informação (TI). Posteriormente, é apresentado o estudo preliminar do processo de entrega de software, relatando o histórico sobre o desenvolvimento de software no instituto, os processos atuais de desenvolvimento e entrega de software e, por fim, uma avaliação dos processos;
2. **Projeto Piloto: O projeto Manhanah** (seção 6.2): nesta seção é apresentado o projeto piloto através de seu relato histórico, descrição e a preparação para o projeto e o relato de experiência da adoção de integração e entrega contínua;
3. **Projeto Irmão: O projeto Cachalote** (seção 6.3): nesta seção é apresentado o projeto irmão através de seu relato histórico, descrição e a preparação para o projeto;
4. **Análise e Resultados** (seção 6.4): nesta seção apresenta os resultados da análise do impacto dessas práticas no processo de entrega de software e na qualidade do código e do produto de software;
5. **Discussão** (seção 6.5): apresenta uma análise dos principais resultados obtidos no estudo.

6.1 AMBIENTE DA PESQUISA: O INSTITUTO FEDERAL DO ACRE

O IFAC é instituição pública federal de educação superior, básica e profissional, pluricurriculares e multicampi, especializada em educação profissional e tecnológica no Acre, criada através da Lei 11.892, de 29 de dezembro de 2008[1], como ação do governo federal para expandir a oferta do ensino profissional.

[1] Disponível em: <https://www.planalto.gov.br/ccivil_03/_ato2007-2010/2008/lei/l11892.htm>

O IFAC iniciou suas atividades acadêmicas no final de julho de 2010, com quatro campi, ofertando cursos técnicos nas modalidades presenciais subsequente, Programa Nacional de Integração da Educação Profissional com a Educação Básica de Jovens e Adultos (PROEJA), superiores em licenciatura e tecnológicos, ofertando apenas para 400 alunos em 9 turmas (IFAC, 2016).

Atualmente, o instituto possui 6 unidades presenciais, sendo que são 5 campi distribuídos nas cidades de Cruzeiro do Sul, Sena Madureira, Tarauacá, Xapuri e em Rio Branco, este possuindo um campus e uma unidade avançada, e 15 polos com ofertas de cursos a distância, alcançando os 22 municípios do estado do Acre. Os campi oferecem cursos de formação inicial e continuada, médio/técnico, tecnológico, licenciaturas, pós-graduação (*lato sensu* e *stricto sensu*) e extensão; em 2017 possuía um total de 5.832 alunos matriculados (PNP, 2019).

Cada campi possui seu próprio setor de TI, responsável por elaborar e implementar projetos, administrar a rede corporativa e oferecer suporte técnico aos usuários (docentes, discentes e técnicos administrativos), subordinados hierarquicamente ao diretor geral do campus e funcionalmente a Diretoria Sistêmica de Gestão de Tecnologia da Informação (DSGTI) da Reitoria.

Dentre as competências da diretoria encontra-se a atribuição de identificar e planejar o desenvolvimento e manutenções de sistemas de informação para o IFAC. Além disso, a DSGTI tem por objetivo desenvolver as atividades de gestão da TI da instituição, bem como, o planejamento, a coordenação, a organização, em nível central, da Tecnologia da Informação afim de alinhar os objetivos, ações e metas às estratégias definidas no Plano de Desenvolvimento Institucional (PDI) e no Mapa Estratégico 2017-2036.

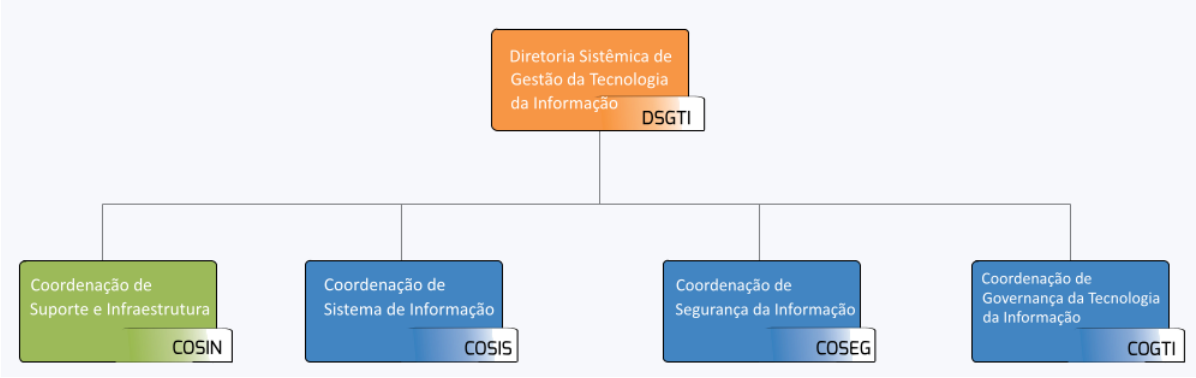
A DSGTI tem como missão prover de forma eficiente e eficaz soluções de TI que possibilitem ao IFAC fornecer educação profissional, cientifica e tecnológica de qualidade no Estado do Acre. O princípio norteador da organização é a visão, que atualmente está descrita como "Ser reconhecido como setor de referência no provimento de soluções de TI no âmbito interno e externo do IFAC".

6.1.1 Estrutura Organizacional

A Figura 12 apresenta a estrutura organizacional da DSGTI/IFAC, que é composta de 1 (uma) direção e 6 (seis) coordenações. Para fins da realização desta pesquisa, iremos focar na coordenação relacionada ao desenvolvimento de sistemas da Diretoria.

- Coordenação de Suporte e Infraestrutura (COSIN): tem como responsabilidade o gerenciamento, controle e planejamento de atividades relacionadas ao suporte técnico;
- Coordenação de Sistema de Informação (COSIS): responsável pela definição, análise, projeto, desenvolvimento, implantação, manutenção, documentação de sistemas de informação dos órgãos de ensino e administrativos da instituição, bem como promover a capacitação destes sistemas para os demais funcionários da instituição;
- Coordenação de Segurança da Informação (COSEG): responsável pelo gerenciamento, controle e planejamento de atividades relacionadas à Infraestrutura, segurança das informações armazenadas da instituição e redes de computadores;
- Coordenação de Governança da Tecnologia da Informação (COGTI): está relacionada ao desenvolvimento de um conjunto estruturado de competências e habilidades estratégicas para profissionais de TI responsáveis pelo planejamento, implantação, controle e monitoramento de programas e projetos de governança. Atuando ainda na gestão de recursos, aquisições de equipamentos, gestão de contratação de TI de forma adequada e demais ações necessárias.

Figura 12 – Estrutura Organizacional DSGTI do IFAC



Fonte: IFAC (2021)[2]

6.1.2 Estudo preliminar do processo de entrega na DSGTI/IFAC

Conforme proposta dessa pesquisa, realizaremos nessa seção o estudo preliminar sobre o processo de entrega e qualidade do produto de software desenvolvido na DSGTI/IFAC,

[2] Disponível em: <https://www.ifac.edu.br/transparencia-e-prestacao-de-contas/organograma>. Acesso em: 25 out. 2021.

incluindo o histórico da área de desenvolvimento e suas respectivas gerências e as tecnologias e processos utilizados atualmente.

6.1.2.1 Histórico

No início do instituto, o desenvolvimento era mais pontual. Com a apenas 3 (três) colaboradores na coordenação de sistemas, o desenvolvimento de produtos de software só ocorria quando não existia solução *open source* ou de terceiros que pudesse suprir a demanda.

Durante a execução dos projetos todos os membros da coordenação eram envolvidos para que qualquer um pudesse dar suporte. Foram desenvolvidas algumas soluções em PHP com PostgreSQL, principalmente para atender demandas de seleções, porém não chegaram a adotar nenhuma metodologia. No entanto, para a criação de sites e outros portais sempre foi utilizado CMS's do mercado.

O principal problema no início era a falta de um sistema acadêmico e administrativo, quando a equipe de desenvolvimento tomou conhecimento de sistemas de seleções de outros institutos federais, mais maduros, realizaram uma cooperação técnica para utilização no IFAC.

Para resolver o problema da falta de um sistema acadêmico e administrativo, o IFAC entrou no uso cooperativo do projeto SIGA-EPCT[3], que tinha como objetivo ser um ERP para atender a todos os institutos. Mas como esse projeto era mantido por bolsas, acabou não avançando e muitos Institutos Federais (IFs) ficaram órfãos.

Quando o projeto SIGA-EPCT não deu certo a equipe de desenvolvimento procurou outras soluções no mercado, sendo elas, o Sistema Unificado de Administração Pública (SUAP)[4] desenvolvido e disponibilizado pelo Insituto Federal do Rio Grande do Norte (IFRN) e os Sistemas Integrados de Gestão (SIGs) desenvolvidos e disponibilizados pela Universidade Federal do Rio Grande do Norte (UFRN). Inicialmente tentaram adotar o SUAP por ser gratuito, mas exigiram na época um corpo técnico grande e com conhecimento em Python[5] e Django[6], e por isso, não aceitaram a cooperação.

A outra solução era adotar o Sistema Integrado de Gestão (SIG) UFRN. Primeiramente

[3] Sistema integrado de gestão acadêmica com a finalidade de automatizar a gestão dos processos institucionais acadêmicos, administrativos e biblioteca.

[4] Sistema desenvolvido em Python com Django pelo IFRN que tem como objetivo a informatização dos processos administrativos e do ensino do Instituto Federal.

[5] *Python* é uma linguagem de programação de alto nível, interpretada, de script, imperativa, orientada a objetos, funcional, de tipagem dinâmica e forte.

[6] *Django* é um *framework* para desenvolvimento rápido para web, escrito em Python, que utiliza o padrão Model-Template-View.

tentaram implantar somente com suporte da UFRN, em 2013, mas como tiveram muitas dificuldades não houveram avanços.

No início de 2014, quando houve troca de reitor(a), resolveu-se contratar uma empresa terceirizada para implantar os SIGs. A busca por um ERP, o apoio fornecido à empresa de implantação e outras demandas locais de serviço, acabou diminuindo a capacidade de desenvolvimento da coordenação. A contratação de novos servidores para compor a equipe da COSIS, a partir de 2015, forneceu um reforço e desafogamento das demandas de sistemas.

Com a contração de pessoal houve uma crescente demanda por desenvolvimento e uma baixa na produtividade, causada pela sobrecarga da equipe, o que ocasionou desperdícios de tempo em retrabalhos, relacionados ao alto número de erros a serem corrigidos e demora para fazer a entrega e a implantação, gerando uma queda de qualidade dos produtos entregues e a perca dos prazos. Por não haver um processo de solicitação de serviços baseados em software, na tentativa de estruturar a área de desenvolvimento de sistemas, foi publicada em agosto de 2017 a Instrução Normativa (IN) Nº 01/2017/DSGTI/CGTI[7] que versa sobre essa solicitação, o que levou a uma melhor alocação dos recursos e um melhor planejamento das atividades.

Atualmente, a COSIS conta com 7 (sete) analistas de TI. Foi definida uma padronização quanto a linguagem de desenvolvimento de seus sistemas, sendo essa o Python utilizando o *framework* Django, e versionando o código no Gitea[8]. Atualmente a coordenação conta com 10 (dez) sistemas próprios, sendo que dois foram entregues recentemente e estão na fase de sustentação, correspondendo à mesma fase que estão os demais sistemas entregues anteriormente.

Como fruto da publicação da IN deu-se início a uma organização e padronização das atividades executadas pela coordenação. Em 2018, foram definidos os atuais processos de negócios de desenvolvimento e sustentação de sistemas e gerência de projetos, com o intuito de aumentar a agilidade na entrega das demandas e melhorar a qualidade do produto entregue. Nas seções a seguir detalharemos esses processos.

[7] Disponível em: https://sisce.s3.ifac.edu.br/media/boletins/Boletim_Ano_VII_41_2017.pdf. Acesso em 29 out. 2020

[8] Serviço git auto-hospedado de código aberto desenvolvido em Go. Disponível em: <https://gitea.io/pt-br/>

6.1.2.2 Processo de Desenvolvimento de Software

O processo de desenvolvimento de um sistema inicia-se com uma solicitação através da abertura de um chamado técnico, contendo uma descrição sucinta do problema e uma comprovação ou justificativa que atestem o alinhamento da solicitação com o PDI e/ou Planejamento Estratégico.

Na segunda fase do processo é realizado um estudo de viabilidade que verificará a existência de sistemas similares no mercado que atendam à demanda, priorizando aqueles que não trarão custos a instituição. Caso não seja encontrada uma solução no mercado, será verificada a possibilidade de atendimento a solicitação, em caso contrário, a decisão será fundamentada.

A terceira fase caracteriza-se pela realização do entendimento do problema através do estudo do negócio, onde é mapeado o fluxo do processo que se deseja sistematizar, além de serem coletados formulários que são usados e legislações internas e externas que impactem o processo. Nesta fase é construído o *product backlog* do produto.

A partir desse ponto o Scrum é utilizado para gerir e planejar o projetos de software. Os papeis são definidos dentre os colaboradores da coordenação, sendo que o papel de *scrum master* geralmente é desempenhado pelo coordenador da COSIS e o *product owner* é desempenhado pelo analista responsável pelos estudos de viabilidade e de negócio, ficando em contato com o cliente do sistema.

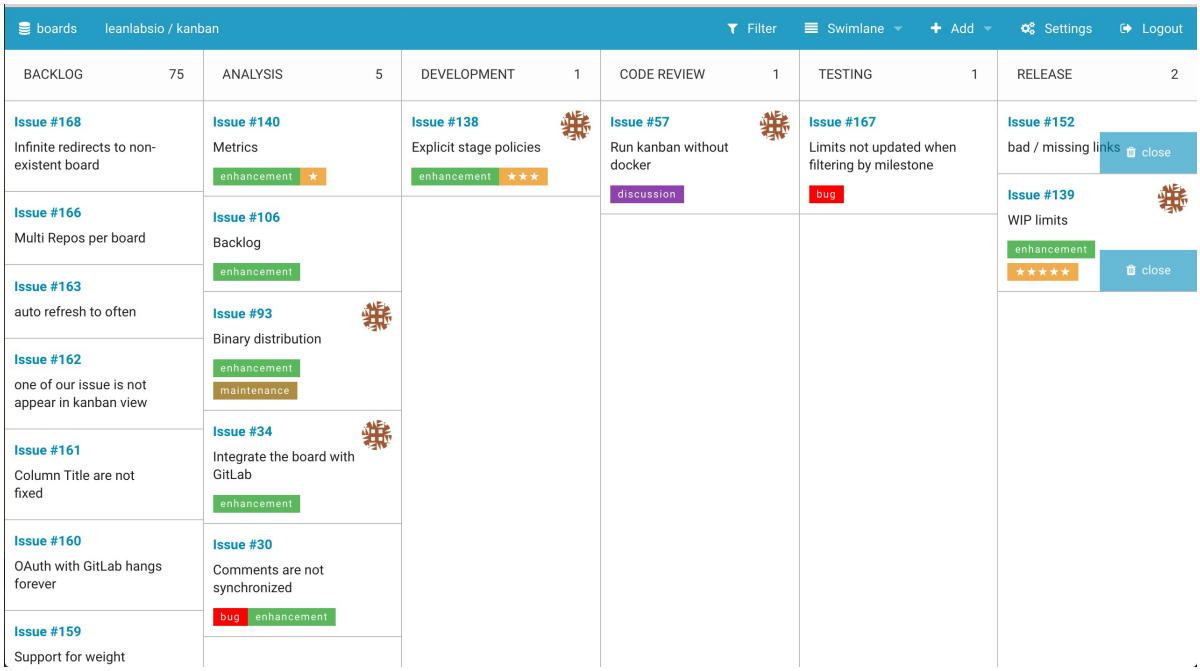
Ao final de cada *sprint*, o produto é disponibilizado para o cliente, seja em um ambiente de homologação ou de produção, dependendo do projeto. Visto que em determinados projetos os clientes preferem que o sistema seja disponibilizado ao público-alvo somente no final das *sprints*.

No caso de manutenções, a solicitação deve conter o detalhamento do problema e o impacto deste no negócio, além de materiais que auxiliem na resolução do problema ou no entendimento da solicitação, tais como imagens (capturas de tela). Diferentemente do desenvolvimento de novos produtos que usa uma customização do Scrum, para gerir softwares em sustentação é usada a metologia Kanban.

A Figura 13 exemplifica o quadro Kanban, o qual foram criadas 6 (seis) colunas afim de organizar os cartões.

[9] Disponível em: <https://twitter.com/leanlabsio>. Acesso em: 25 nov. 2020.

Figura 13 – Exemplo de um Quadro de Tarefas Kanban



Fonte: Página do LeanLabs no Twitter[9]

6.1.2.3 *Processo de Entrega de Software*

Na COSIS, há uma padronização quanto ao processo de entrega indiferentemente da fase em que o projeto esteja, seja desenvolvimento de um novo produto ou sustentação, o fluxo de entrega do produto é o mesmo.

Durante a implementação da funcionalidade o desenvolvedor a testa localmente de forma manual e não tendo sido encontrado nenhum erro ele o commita e faz o *push* para a *branch master* do repositório central no Git.

Para realizar a primeira implantação do produto de software em um servidor de homologação ou produção, o analista da COSIS verifica se há alguma máquina que possa ser utilizada para a implantação. Caso contrário, o analista solicita, através de um chamado técnico, uma Máquina Virtual, em inglês *Virtual Machine* (VM) para a COSEG, descrevendo o serviço que irá rodar na máquina, especificando o sistema operacional e, caso necessário, a quantidade de memória e o número de processadores.

Com a VM criada, o desenvolvedor irá configurá-la para poder realizar a implantação. Depois de instaladas todas as dependências e serviços, o código da aplicação é enviado para o servidor via *SSH File Transfer Protocol* (SFTP), em seguida, é feita a configuração do projeto

e dos serviços, como por exemplo, o Nginx[10]. Para a implantação de uma aplicação Django, os seguintes passos são executados: (i) atualizar o código fonte; (ii) instalar dependências da aplicação; (iii) aplicar as migrações do banco de dados; (iv) coletar arquivos estáticos; (v) reiniciar servidor de aplicação e serviços associados; e (vi) reiniciar servidor web.

Por fim, caso seja necessário a criação de um domínio ou subdomínio para a aplicação, é aberto um novo chamado para a COSEG, informando se a entrada de domínio funcionará interna e/ou externamente e para qual servidor ela irá apontar.

6.1.2.4 Avaliação dos processos

Apesar da proposta de um desenvolvimento ágil, com papeis e áreas definidas os processos em questão não conseguiram atingir esse objetivo. Após 3 (três) anos de implantação a coordenação desenvolveu 2 projetos: a Plataforma de Eventos Cachalote e o Sistema de Regulamentação de Atividades Docentes (SISRAD)

Ao ser analisada a documentação dos sistemas, o gerenciamento de projetos e métricas utilizadas, constatamos que na coordenação não há coleta de métricas e nem uma padronização na documentação dos sistemas.

As métricas coletadas são relacionadas ao sistema de chamados técnicos (GLPI)[11], portanto as métricas existentes são: tempo de solução (*lead time*), satisfação do usuário, quantidade de chamados e etc. No entanto, a equipe não recategoriza o chamado, nem define sua ordem de prioridade, portanto há chamados categorizados erroneamente pelo requerente.

Analisamos o *lead time* e o número de chamados relacionados ao desenvolvimento ou sustentação de sistemas. O *lead time* foi coletado do sistema de chamados técnicos do qual os chamados foram exportados para uma planilha eletrônica. Foi realizada uma filtragem dos chamados, sendo retirados os chamados que não eram relacionados a correção de erros/*bugs* e a adição/alteração de funcionalidade nos sistemas desenvolvidos pela equipe; o cálculo do *lead time* teve como base a data de abertura e a data que o chamado foi dado como solucionado (implantado em produção).

Primeiramente foi coletado o *lead time* dos chamados referentes à correção de erros/*bugs* na aplicação. No total foram registrados 225 (duzentos e vinte e cinco) chamados desde o 2019, ano esse em que foi configurado, nos sistemas desenvolvidos pela equipe, a abertura

[10] *Nginx* é um servidor web *open source*.

[11] Sistema de código aberto para o gerenciamento de ativos de TI. Disponível em: <https://glpi-project.org/pt-br/>

automática de chamados técnicos de erros críticos dos sistemas. Para o processo de correção de um problema (*bugs*) crítico em uma aplicação o *lead time* médio é de 7 (sete) dias. Já o *lead time* médio dos chamados para implementar uma nova funcionalidade é de 10 dias.

Além da ferramenta de chamados, foram analisados os repositórios Git dos projetos desenvolvidos pela equipe e verificou-se que os sistemas não possuem testes automatizados, ou seja, a cobertura de testes é inexistente.

No Gitea, não estão versionados os arquivos de configuração que devem ser usados para implantar o projeto. Constatando-se que não há rastreabilidade das modificações feitas no código, através de *issue tracker* ou um sistema de gerenciamento de projetos, que contenha o detalhe da alteração ou do problema, quem solicitou, a data do início da implementação, e da implantação e etc. Portanto, a rastreabilidade restringe-se as mensagens do *commit* e quem realizou a mudança. O que impossibilita a coleta de algumas métricas, como por exemplo, velocidade, tempo de ciclo, cobertura, throughput (taxa de transferência).

O processo de entrega do produto é realizado de forma manual pelo analista responsável e não repetível. Portanto, observa-se que os antipadrões comuns de entrega de versão descritos por Humble e Farley (2014) ocorrem na COSIS, sendo eles: implantar software manualmente; e gerência de configuração manual dos ambientes de produção.

Mesmo com a adoção dos processos ágeis de desenvolvimento a entrega do produto de software é custosa e não repetível, devido ao excesso de erros que ocorrem durante o processo de *deploy*, acarretando em correções frequentes, gerando uma demora para fazer a entrega e a implantação sem garantia da qualidade do produto, devido a um alto número de erros que são reportados tardiamente, gerando atrasos e consequentemente a perca dos prazos além de uma baixa produtividade do setor.

Assim, a integração e a entrega contínua surgiram como uma alternativa viável para a solução dos problemas apresentados nos processos da COSIS, principalmente por propor uma redução do tempo de lançamento do produto, além de segurança e qualidade nas entregas. Além disso, alguns integrantes da equipe já demonstravam interesse na utilização dessas práticas, o que facilitou a direção da coordenação a aceitar as novas práticas de desenvolvimento.

Nas subseções a seguir detalharemos a adoção desses métodos na COSIS/DSGTI, através do estudo de caso de um projeto que foi construído utilizando essa nova abordagem.

6.2 PROJETO PILOTO: O PROJETO MANHANAH

Dentre os projetos na fila para construção escolheu-se o Manhanah, por ser considerado um projeto estratégico para o IFAC, devido ao fato de possibilitar uma maior visibilidade do público externo ao Plano Individual de Trabalho (PIT) e ao Relatório Individual de Trabalho (RIT) e uma melhor gerência do processo à alta gestão.

O desenvolvimento deste projeto está embasado na Resolução CONSU/IFAC Nº 001, de 09 de janeiro de 2019[12] que versa sobre aprovação das normas sobre a Regulamentação das Atividades Docentes (RAD) dos (as) professores (as) do Ensino Básico, Técnico e Tecnológico do instituto e na Nota Técnica nº 01/2017/PROEN/RIFAC que trata sobre orientações para registro da conversão de horas-aulas em minutos para hora de 60 minutos, considerando critérios de arredondamento e padronização do número correspondente a horas, além de orientações do Tribunal de Contas da União (TCU) e da Controladoria Geral da União (CGU) o projeto vem com o objetivo de possibilitar uma maior visibilidade do público externo ao PIT e ao RIT e uma melhor gerência do processo à alta gestão.

6.2.1 Descrição do projeto

Objetivo do projeto Manhanah era desenvolver uma solução sistêmica para automatizar o processo de entrega do PIT e RIT. Este projeto surgiu da necessidade de haver uma maior publicidade dos documentos, tornando o IFAC mais transparente ao público externo. O sistema deve facilitar o preenchimento do PIT/RIT por parte do docente que deve entregá-lo a cada final de semestre, além de ajudar no gerenciamento para tomada de decisões, maior transparência do processo e controle, bem como torná-lo auditável, garantindo maior confiabilidade e com segurança.

6.2.2 Preparação para o projeto

A equipe formada foi composta pelo coordenador da COSIS desempenhando o papel de *scrum master* e 2 (dois) analista de TI . O papel de *product owner* foi desempenhado por um dos desenvolvedores, que foi responsável por conhecer o negócio a fundo. O Quadro 11

[12] Disponível em: <https://sisce.s3.ifac.edu.br/media/boletins/Boletim_Ano_IX_02_2019.pdf>. Acesso em: 14 abr. 2022

apresenta o perfil dos colaboradores do projeto Manhanah.

Quadro 11 – Perfil dos colaboradores do projeto Manhanah

Colaborador	Faixa Etária	Tempo de Instituição (anos)	Experiência com desenvolvimento e implantação (anos)	Cargo
Coordenador	30-34 anos	5	14	Analista de TI
Desenvolvedor 1	25-29 anos	5	8	Analista de TI
Desenvolvedor 2	35-39 anos	6	16	Analista de TI

Fonte: Elaborado pelo autor (2022)

Além da utilização do Scrum para gerir o projeto, as seguintes práticas XP foram selecionadas:

- Padronização do código, prática já utilizada na COSIS/DSGTI;
- Propriedade coletiva do código, também já utilizada pela coordenação;
- Testes, construção de testes automatizados para validar as funcionalidades;
- Refatoração;
- Integração contínua, prática utilizada parcialmente pela coordenação, pois só o código-fonte da aplicação é armazenado em um repositório único, ao qual todos os desenvolvedores possuem acesso, no entanto, no momento de integração do código não são executados testes devido a cultura organizacional.
- *Design* simples, com vistas a evitar o desperdício no desenvolvimento;
- Entregas curtas, a cada duas semanas, conforme previsto;
- Jogo do planejamento, para auxiliar na priorização e na estimativa das atividades; e
- Cliente presente, para melhor entendimento do negócio e *feedback*.

A prática "Progamação em Pares" do XP foi excluída devido ao número de colaboradores da COSIS que poderiam ser designados para esse projeto, visto o números de demandas da coordenação.

6.2.3 Descrição da Experiência

Conforme relatado no capitulo 5, a primeira versão da abordagem da adoção de integração e entrega contínua envolveu as evidências coletadas na revisão sistemática. Nessa seção faremos um relato de experiência do uso dessa abordagem para adotar integração e entrega contínua em um projeto piloto da DSGTI, denominado "Manhanah".

O projeto "Manhanah" teve início com a abertura do chamado técnico solicitando o desenvolvimento de um sistema, contendo os embasamentos legais. No estudo de viabilidade do projeto, foram encontradas soluções no mercado, como o módulo do SUAP, no entanto, verificou-se que estas não atenderiam as necessidades.

Na fase de entendimento do problema o analista responsável pelo projeto, fez um estudo do negócio no qual, foram criados os fluxos do processo através da notação Modelo e Notação de Processos de Negócio, em inglês *Business Process Model and Notation* (BPMN)[13], que serviram de insumo para a produção do *Product Backlog*. A medida que as histórias eram refinadas, as mesmas eram cadastradas no repositório do projeto do Gitea, como *issues*.

A fase de desenvolvimento do projeto teve início em maio de 2019, no entanto a adoção do CI/CD teve um atraso, devido a equipe não ter uma cultura e nem experiência no desenvolvimento de testes automatizados. Para sanar esse problema, foi ofertado um treinamento para capacitar o time nessa prática, possibilitada pela paralisação do projeto, ocasionada por férias de membros do projeto.

Após o reinício do projeto, em outubro do mesmo ano, iniciou-se o estudo sobre a adoção do CI/CD. Primeiramente, foi observada se a equipe executava as três práticas necessárias para adotar CI/CD, estabelecidas por Humble e Farley (2014). Abaixo, estão descritas as práticas relacionadas com o fluxo de trabalho da equipe de desenvolvimento:

1. **Controle de versão**: o projeto deve estar versionado em um repositório único e tudo o que for necessário para criar, instalar, executar e testar sua aplicação. A prática já era utilizada pela equipe, no entanto, os arquivos de configuração dos ambientes de homologação e de produção não eram versionados;

2. **Processo automatizado de compilação**: devido ao uso da linguagem interpretada Python com o *framework* Django não se faz necessário automatizar o processo de com-

[13] *BPMN* é uma notação gráfica, que tem por objetivo prover recursos e elementos para modelar (desenhar) processos de negócio.

pilação;

3. **Aceitação da equipe**: A equipe estava motivada a adotar as práticas CI/CD, devido as dificuldades encontradas para realizar a implantação de forma manual e a demora para entregar da aplicação no ambiente de teste/homologação.

Após a verificação dos pré-requisitos para a adoção do CI/CD, foi mapeado o fluxo de entrega de uma funcionalidade do sistema, tendo como atividade inicial a análise da funcionalidade e terminando com esta disponível em produção. Para a construção do fluxo de valor foi utilizado o processo de entrega da correção de um defeito realizado pela COSIS/DSGTI, no qual foi considerado somente tempo gasto em cada etapa, sendo desconsiderada a espera entre as etapas devido a grande variação de tempo encontrada durante a construção desse fluxo, ocasionada por demandas externas ao projeto.

A partir do fluxo de valor da entrega de uma funcionalidade mapeado, partimos para as definições do fluxo de trabalho do controle de versão e para a escolha e configuração das ferramentas.

Como fluxo de trabalho do sistema de controle de versão foi adotado o Desenvolvimento Baseado no Trunk, em inglês *Trunk-based development* (TBD), ou em português desenvolvimento baseado em troncos, que estabelece que os desenvolvedores trabalhem no código em uma única ramificação chamada "tronco", evitando a criação de ramificações de desenvolvimento (para mais detalhes veja na subseção 5.2.2). Embora essa prática desencoraje a criação de novas ramificações, para equipes maiores ou com grande fluxo de confirmações, ela é melhor executada com ramificações de recursos de curta duração de uma pessoa e fluxo *pull request* antes de "integrar" ao tronco.

Foram adotadas algumas práticas relacionadas ao controle de versão que deviam ser seguidas pelos membros da equipe. Os desenvolvedores deveriam realizar *commits* ligados a uma tarefa, sendo que deveriam ser feitos pelo menos um *commit* por dia na *branch master*, além de serem pequenos.

Para escolhermos as ferramentas, levamos em conta ferramentas de código aberto e livre que pudessem ser instaladas na infraestrutura da diretoria. Para gerenciar os repositórios Git, optou-se em permanecer com a ferramenta em uso, no caso, o Gitea. O Gitea permite que sejam criadas organizações, as quais podem ser proprietárias de todos os repositórios, conforme a prática do XP de propriedade coletiva do código, portanto, na configuração da ferramenta, foi criada a organização "COSIS".

6.2.3.1 Construção do Pipeline

Para a implantação do pipeline de integração e entrega contínua, foi escolhido o servidor de automação Jenkins[14], cujas funcionalidades podem ser estendidas, permitindo a integração com diversas ferramentas através da instalação de plug-ins. O Jenkins agiliza o *feedback* aos desenvolvedores, pois a cada commit e integração ele é acionado, executando uma série de testes e verificações do código.

Na configuração do servidor Jenkins foram instalados os plugins sugeridos e os plugins *LDAP Plugin*, para permitir a autenticação através do *Active Directory* (AD), *Blue Ocean*, *Cobertura Plugin*, para possa ser publicado o relatório de cobertura de código, *Gitea Plugin*, para permitir a integração com o Gitea, *Pipeline*, *Build Timeout*, para que possam ser colocados tempos limites em algum estágio do pipeline, *Publish Over SSH*, para implantação via SSH, *SonarQube Scanner for Jenkins*, para permitir a integração com o Jenkins e *Violations plugin*, para publicar os relatórios da verificação dos linters.

A integração do Jenkins com o Gitea foi realizada através da criação de um projeto ou *job*[15] do Jenkis e com base na "*Organization*" definida no Gitea, ou seja, para cada organização existente deve ser realizada uma nova configuração. Porém, da forma como é configurado o projeto Jenkins, caso um repositório do Gitea "*Organization*" deseje implantar um pipeline, basta criá-lo usando o *Jenkinsfile*.

A configuração do pipeline de integração e entrega contínua foi realizada através da criação de um pipeline declarativo usando o *Jenkinsfile* (disponível por completo no Apêndice F), onde as primeiras etapas criadas foram o *checkout* do código e a instalação de dependências da aplicação. Durante a configuração do passos de instalação das dependências, obtivemos inúmeros erros que faziam este passo falhar, devido as dependências das bibliotecas da aplicação com pacotes do sistema operacional. Para solucionar o problema, foi configurado o pipeline de forma que as etapas fossem executadas em um contêiner Docker[16] configurado através de um *Dockerfile*, no qual foram instalados os pacotes necessários.

Inicialmente a verificação do repositório do código ocorria de 5 (cinco) em 5 (cinco) minutos, porém verificamos que essa configuração poderia gerar problemas futuros devido

[14] *Jenkins* é um servidor de automação de código aberto que pode ser usado para automatizar tarefas relacionadas à construção, teste e entrega ou implantação de software. Disponível em: <https://www.jenkins.io/>

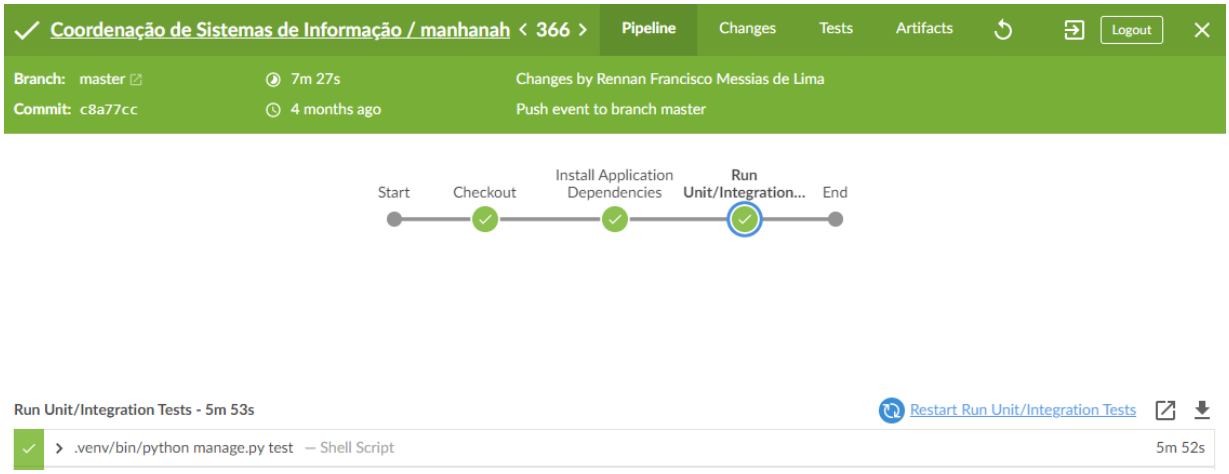
[15] *Projeto* ou *Job* é uma descrição do trabalho configurada pelo usuário que o Jenkins deve realizar, como construir um software, etc.

[16] Disponível em: <https://www.docker.com/>

ao crescimento de projetos. Portanto, optamos por configurar o servidor de CI para verificar o repositório de código somente quando houvesse operações como *push*, *pull request* e/ou criação de versões, através do *webhook*.

A etapa seguinte a ser configurada foi a de execução dos testes unitários e de integração, que devido ao uso do *framework* Django facilita a construção destes testes, sendo executados no mesmo comando podendo consumir muito tempo do pipeline. Na Figura 14 pode ser visto que a execução dos testes demorava cerca de 6 (seis) minutos. Para resolver esse problema, tentamos paralelizar a execução destes testes, porém quando um teste não passava a equipe demorava mais para corrigir, então retiramos a execução em paralelo, diminuindo o tempo de correção da quebra do pipeline. Por fim, descobrimos que o problema da demora da execução dos testes decorria de uma biblioteca para gerar tabelas de auditória, portanto, foi configurada para ser ativada somente em produção o que reduziu o tempo de execução dos testes de 6 (seis) para 2 (dois) minutos.

Figura 14 – Etapas do pipeline configuradas inicialmente



Fonte: Elaborado pelo autor (2022)

A construção do pipeline foi paralisada devido a equipe está tendo dificuldades com a definição das próximas etapas, mesmo com o fluxo de valor definido. Então, a equipe se reuniu e definiu as etapas do pipeline de integração e entrega contínua, como pode ser visto na subseção 6.2.4.2.

Com o pipeline conceitual, foi criada a etapa de análise estática do código e configurada para conter dois estágios executando em paralelo. Um estágio é a execução de bibliotecas Python para verificar a padronização de codificação e o outro é a análise do SonarQube[17], que

[17] *SonarQube* é uma plataforma de código aberto desenvolvida pela SonarSource para inspeção contínua da qualidade do código. Disponível em: <https://www.sonarqube.org/>

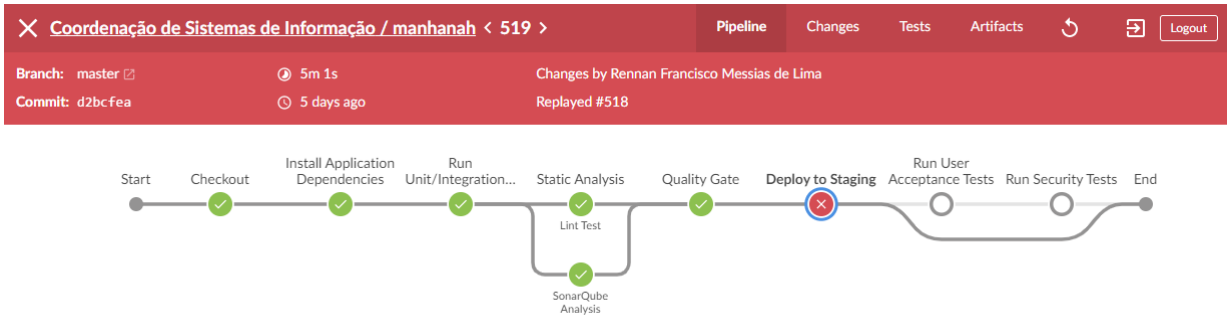
além de realizar uma análise da qualidade do código faz uma análise estática de segurança e fornece diversas métricas de segurança e qualidade.

Ao integrarmos o SonarQube com o Jenkins e executarmos o pipeline o escaneamento da aplicação pelo Sonar não era realizado, embora pudesse ser executado por linha de comando diretamente no servidor. Observamos então, que o problema estava na construção do *script* de configuração do pipeline, pois como estava configurado para usar o container Docker especificado no *Dockerfile* em todas as etapas, este não encontrava o programa de escâner do SonarQube. Portanto, reconfiguramos o pipeline para que em cada etapa pudesse ser definido um agente de execução, com isso o escâner passou a ser executado em um agente definido pelo Jenkins.

Após ser executada a etapa de análise estática, o Jenkins aguarda um retorno do SonarQube para saber se o novo código passou nos critérios de qualidade estabelecidos. Caso o código passe nos critérios de qualidade, ocorre a implantação da aplicação no ambiente de homologação e testes (*Staging*).

A automatização da tarefa de implantação foi realizada primeiramente através da construção de um *Shell script*. Foram cadastradas as chaves SSH dos ambientes de homologação e produção no Gitea como "Chaves de *Deploy*", sem permissão de escrita no repositório central. Com a criação desse *script*, verificamos que a tarefa de implantação reduziu de uma média de 4 horas para 3 minutos. No entanto, ocorriam alguns erros durante a execução do pipeline, ocasionando a interrupção de sua execução e o Jenkins apresentava visualmente em qual passo o erro aconteceu, como ilustrado na Figura 15, além de notificar os membros da COSIS por e-mail.

Figura 15 – Erro durante a execução do pipeline



Fonte: Elaborado pelo autor (2022)

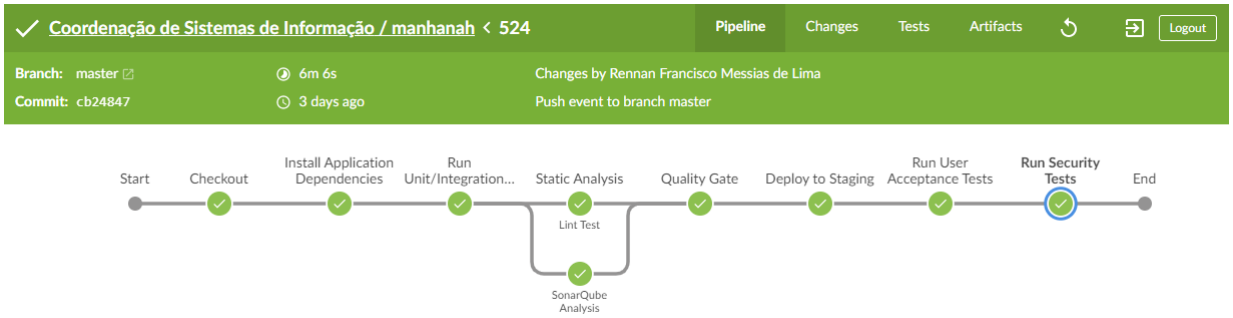
Em decorrer de alguns erros com as implantações via *Shell scripts*, a equipe passou a utilizar

a biblioteca Fabric[18] para automatizar a implantação da aplicação nos ambiente de *Staging* e Produção. O Fabric facilitava a manutenção do script de implantação, devido a abstração Python para executar comandos shell remotamente via SSH, fornecida pela biblioteca.

Com a aplicação disponível no ambiente de *Staging*, são executados os testes automatizados de aceitação do usuário. Para automatizar esses testes foi utilizada a biblioteca *behave-django*[19] juntamente com a biblioteca *selenium*[20]. O *behave-django* fornece uma forma de criar testes BDD integrado ao Django, através dos cenários de teste escritos em linguagem natural que utiliza palavras-chaves para descrever os passos que devem ser executados. Para executar esses testes no navegador utilizamos a biblioteca *selenium* que fornece recursos para automatizar diversas ações, como digitação em campos de texto, cliques em botões, *submits* de formulários, entre outras.

A última etapa do pipeline é a verificação automatizada de segurança, fornecida pelo próprio *framework* Django, afim de revisar as configurações da aplicação antes da implantação em produção. Por fim, foi configurada uma etapa de limpeza que sempre será executada, independentemente de a execução do pipeline quebrar ou não, e outra de notificação em caso de quebra do pipeline. A Figura 16 apresenta todas as etapas da pipeline após a execução.

Figura 16 – Etapas do pipeline após a execução



Fonte: Elaborado pelo autor (2022)

[18] *Fabric* é uma biblioteca Python projetada para executar comandos shell remotamente por SSH. Disponível em: <http://www.fabfile.org/>

[19] *behave-django* é uma biblioteca Python para integrar Behave BDD com Django. Disponível em: <https://behave-django.readthedocs.io/>

[20] *selenium* é uma biblioteca Python para escrever testes funcionais/de aceitação usando Selenium WebDriver. Disponível em: <https://selenium-python.readthedocs.io/>

6.2.3.2 Entrega de Versão

O processo de entrega inicia-se com a identificação da versão em um arquivo de configuração, seguido pelo *commit* que tem como mensagem o número da versão. O Jenkins executa o pipeline referente ao *commit* da versão, em caso de falha o processo é interrompido, caso contrário, é criada a tag da versão, realizada a implantação da respectiva tag e por fim são executados testes de fumaça (smoke tests), para verificar o sistema em funcionamento.

A entrega do produto para os usuários foi divida em duas etapas. Na primeira etapa foi entregue o Mínimo Produto Viável, em inglês *Minimum Viable Product* (MVP), que para este projeto significou disponibilizar uma versão da aplicação contendo todo o fluxo de criação, validação e homologação do PIT. O MVP serviu como avaliação do sistema e de sua aceitação, tendo em vista a influência do mesmo no trabalho dos docentes do instituto.

Ao final da última *sprints* foi entregue o Produto Mínimo Vendável, em inglês *Minimal Marketable Product* (MMP), que para este projeto significou disponibilizar uma versão da aplicação contendo o fluxo de criação, validação e homologação do PIT e do RIT, marcando o fim do desenvolvimento do produto. Na reunião de retrospectiva final foi aberta uma discussão sobre a nova abordagem, a qual foi aceita como positiva pelo time. Portanto, o desenvolvimento do produto, serviu como aprendizado da utilização da nova abordagem.

6.2.4 Processo de Adoção de Pipeline de CI/CD

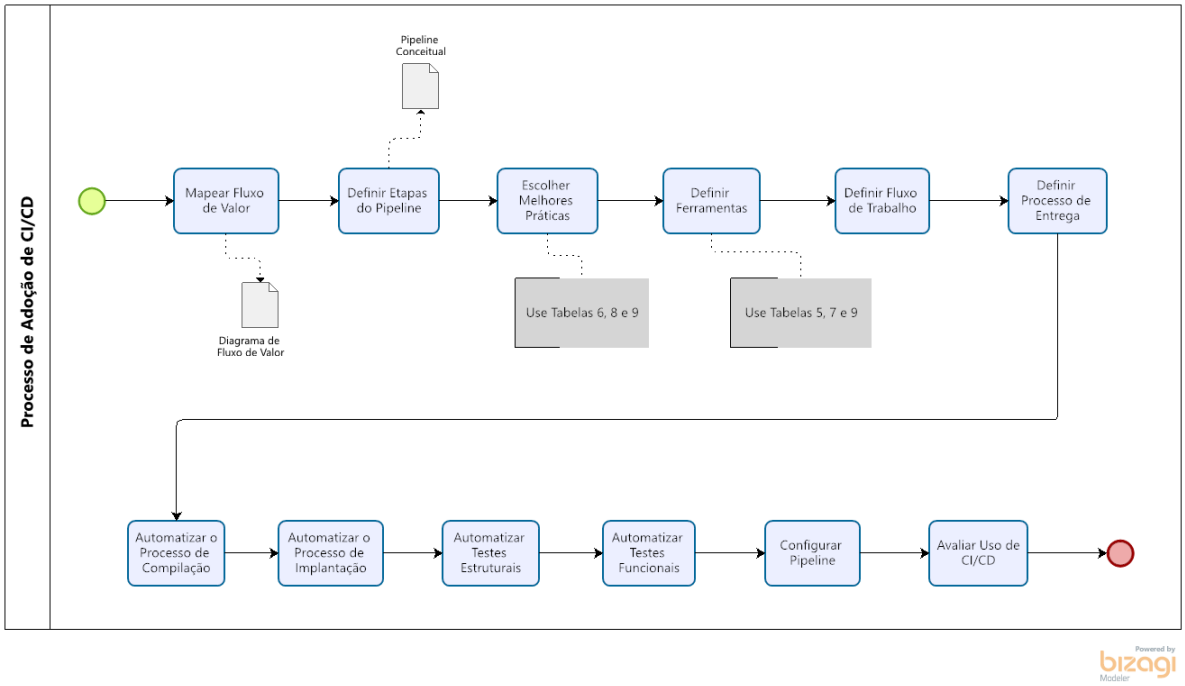
Após a conclusão do projeto piloto, o processo de adoção preliminar foi atualizado, de forma a contemplar as experiências obtidas durante a execução desse projeto. Portanto, foi adiantada a tarefa de definição das etapas do pipeline, modificada a ordem da escolha e definição das melhores práticas e ferramentas e foram acrescentadas 3 (três) etapas: (i) definição do fluxo de trabalho; (ii) definição do processo de entrega; e (iii) avaliação do uso do CI/CD. O processo atualizado e as novas tarefas são ilustradas na Figura 17, e detalhadas nas seções a seguir.

6.2.4.1 Mapear Fluxo de Valor

O mapeamento do fluxo de entrega de uma funcionalidade do sistema, tem como atividade inicial a análise da funcionalidade e terminando com esta disponível em produção.

A Figura 18 apresenta o fluxo de valor do processo de entrega da correção de um defeito

Figura 17 – Processo de Adoção de Pipeline de CI/CD Atualizado



Fonte: Elaborado pelo autor (2022)

realizado pela COSIS/DSGTI, no qual, foi considerado somente tempo gasto em cada etapa, sendo desconsiderada a espera entre as etapas devido a grande variação de tempo encontrada durante a construção desse fluxo, ocasionada por demandas externas ao projeto.

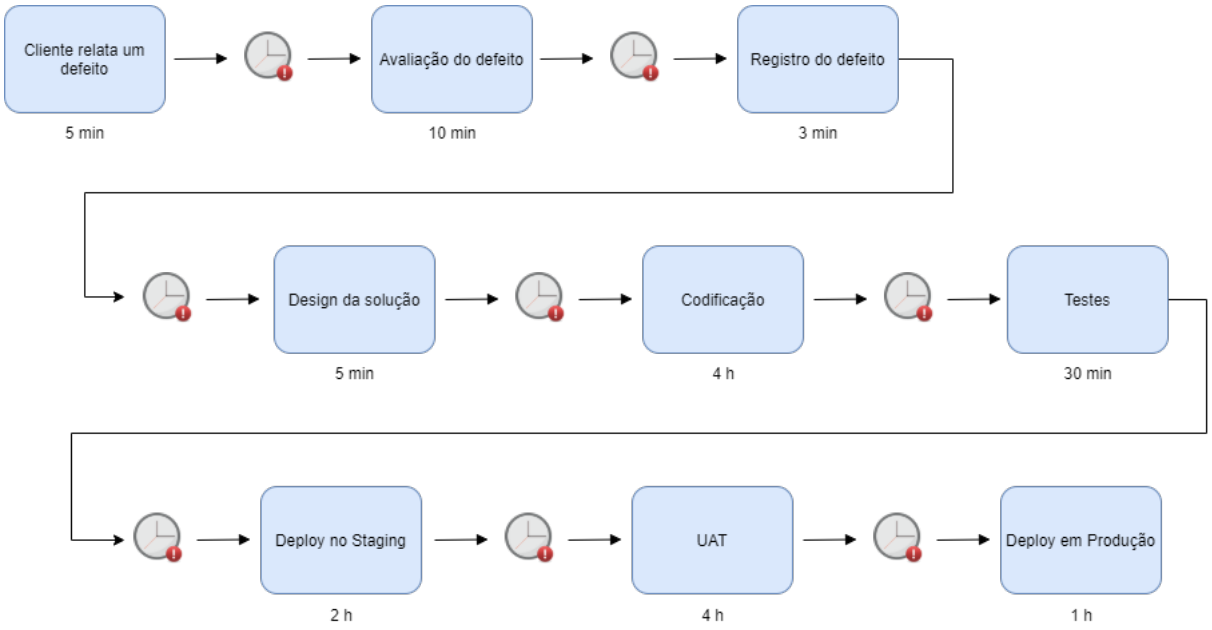
Com relação ao fluxo de valor do processo de entrega de uma funcionalidade durante a etapa de desenvolvimento, leva-se em consideração o digrama apresentado na Figura 18, no qual o seu início é alterado para a tarefa "Design da solução".

6.2.4.2 Definir Etapas do Pipeline

As etapas do pipeline são definidas de acordo com o diagrama de fluxo de valor. Após listadas todas as etapas que o pipeline terá, é criado um diagrama, como forma de documentação, para futuramente seja usado na configuração do pipeline real.

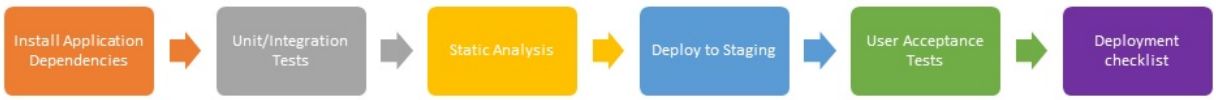
O diagrama do pipeline do projeto piloto, apresentado na Figura 19, contem as etapas de construção (*build*), testes unitários, análise da qualidade do código, implantação em um ambiente de homologação, execução dos testes de aceitação e por fim um checagem de implantação em produção.

Figura 18 – Diagrama de Fluxo de Valor da Entrega do Processo de Entrega



Fonte: Elaborado pelo autor (2022)

Figura 19 – Pipeline CI/CD Desenvolvido para este Estudo



Fonte: Elaborado pelo autor (2022)

6.2.4.3 Escolher Melhores Práticas

Nessa etapa do processo, a equipe deve se reunir e definir a metodologia de desenvolvimento. Com base nessa escolha, definir as práticas a serem utilizadas durante o desenvolvimento.

Podem ser utilizadas práticas de diversas metodologias aliadas a metodologia base, como por exemplo, no projeto piloto foi escolhido o Scrum como metodologia base com práticas do *Extreme Programming* (XP). A seguir são listadas as práticas, adicionais as listadas na subseção 6.2.2, utilizadas durante o desenvolvimento do projeto piloto:

- MP1. Adoção de Práticas Ágeis (Scrum com algumas práticas do XP)
- MP2. Padronizar o fluxo de trabalho no Sistema de Controle de Versões
- MP3. Mudanças no esquema do banco de dados no sistema de controle de versão

- MP4. Todos os Commits estão ligados às tarefas
- MP5. Commit de código com mais frequência
- MP6. Commits pequenos
- MP7. Trabalhas em pequenos lotes
- MP8. Refatoração
- MP10. Análise Estática de Código Automática
- MP11. Coleta de métricas de qualidade
- MP12. Testes automatizados
- MP13. Definir Estratégias de testes
- MP14. Melhorar a atividade de testes: para o projeto piloto foram usados os seguintes tipos de testes: testes de unitários; testes de integração; teste de cobertura; teste de regressão; testes de aceitação e testes de segurança
- MP17. *Build* automatizado
- MP18 Implantação automatizada

6.2.4.4 Definir Ferramentas

Durante a escolha das ferramentas foram utilizados os seguintes critérios, ser de código aberto e que pudessem ser instaladas na infraestrutura da diretoria. O Quadro 12 mostra as ferramentas utilizadas na adoção deste pipeline de integração e entrega contínua, explicando o motivo de sua escolha.

Com o uso do SonarQube, muitas métricas puderam ser obtidas de forma automática. Entretanto, algumas métricas tiveram a necessidade de serem coletadas manualmente. Para isto, a estratégia utilizada foi armazenar os dados na *issues* do repositório e posteriormente registrá-las em uma planilha eletrônica.

Quadro 12 – Ferramentas Utilizadas

Finalidade	Nome	Motivo
Repositório de Código	Gitea	Ferramenta *open source*, que já era utilizada pela COSIS. Posteriormente foi abandonada e seus repositórios migrados para o GitLab
	GitLab	Foi migrado para o GitLab devido ao IFAC a este ter mais funcionalidades
Compilação	Makefile	Utilizado para abstrair a configuração do ambiente de desenvolvimento
	Docker	Utilizado como base para criação contêiner onde o Jenkins executava as etapas do pipeline
	Fabric	Utilizado para realizar a implantação tanto em homologação quanto em produção
Servidor CI	Jenkins	Ferramenta *open source*, cujas funcionalidades podem ser estendidas, permitindo a integração com diversas ferramentas através da instalação de plug-ins
	GitLab CI	Foi migrado para o GitLab CI devido a inúmeros problema de compatibilidade do Jenkins com o Gitea e com o próprio GitLab
Testes Unitários e Testes de Integração	Django	Os testes do Django usam um módulo de biblioteca padrão Python *unittest* baseada no JUnit. O Django fornece a criação bancos de dados em branco separados para execução dos testes, além de um cliente de teste para recuperar páginas da Web
Análise Estática	SonarQube	Ferramenta *open source*, sendo a mais citada na revisão sistemática
Teste de Aceitação	Behave e Selenium	O Behave juntamente com o Selenium automatizam os testes de comportamento e de aceitação do usuário nos navegadores
Mudança do Esquema de Banco de Dados	Django	O Django fornece uma forma de que as mudanças feitas nos modelos sejam propagadas para o esquema de banco de dados
Implantação Automatizada	Fabric	Biblioteca Python para executar comandos shell remotamente por SSH

Fonte: Elaborado pelo autor (2022)

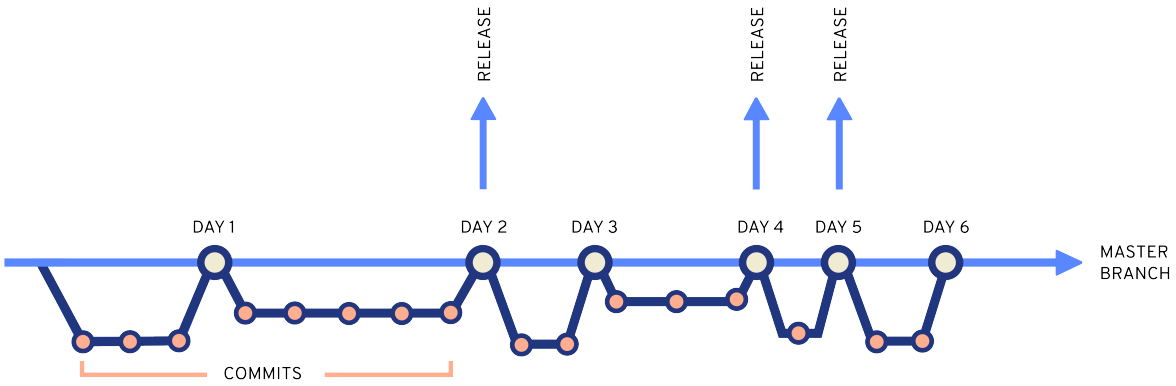
6.2.4.5 Definir Fluxo de Trabalho

A padronização do fluxo de trabalho dos sistemas de controle de versão é indispensável, pois adotar o versionamento de código sem tem ter um fluxo de trabalho definido pode dificultar a utilização de CI/CD. Existem diversos fluxos de trabalho de desenvolvimento usando o Git, dentre eles o TBD, *Git Flow* e o *Git Feature Branch Workflow* e etc.

Como fluxo de trabalho de desenvolvimento usando o Git foi adotado o TBD, que estabelece que os desenvolvedores trabalhem no código em uma única ramificação chamada "tronco", evitando a criação de ramificações de desenvolvimento.

O TBD se assemelha com *Git Feature Branch Workflow*, em português fluxo de trabalho de *branch* de recursos do Git, que estabelece que todo o desenvolvimento de recursos deve ocorrer em uma ramificação dedicada em vez da ramificação principal. Sendo que a ramificação de desenvolvimento só será integrada a ramificação principal após a conclusão do desenvolvimento do recurso. Já o TBD, encoraja ter várias ramificações de recursos pequenos e de curta duração e mesclá-las na ramificação principal o mais rápido possível. A Figura 20, apresenta o fluxo do TBD, parecido com o que foi adotado no projeto piloto, com implantação em produção sendo marcadas com tags.

Figura 20 – Fluxo de Trabalho TBD



Fonte: Obvious Engineering (2020)[21]

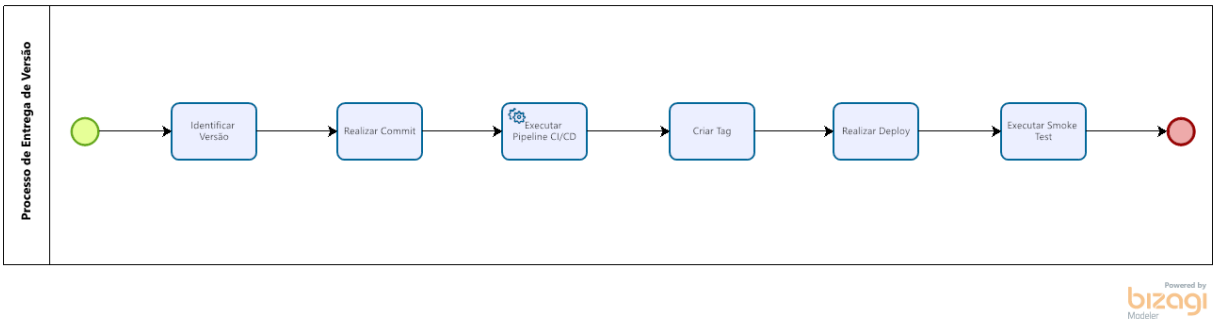
6.2.4.6 Definir Processo de Entrega

Durante a entrega de uma versão introduz-se o risco de vulnerabilidades, problemas, *bugs* e software sem desempenho acontecerem em produção. Portanto, definir uma estratégia de liberação que funcione reduz esses riscos. Para o projeto piloto, escolhemos uma adaptação da *feature flags*, no qual os novos recursos ficam escondidos dos usuários, no entanto, não escolhemos um mecanismo de *roll-back*, porém para a adoção da implantação contínua essa prática se faz necessária.

[21] Disponível em: <https://engineering.obvious.in/release-engineering/trunk-based-development>. Acesso em: 13 set. 2022.

A Figura 21 apresenta o processo BPMN da entrega de versão definido para o projeto piloto. A implantação em produção é realizada a partir da criação de uma tag no GitLab. Nesse momento é executado o pipeline completo como forma de assegurar que a versão que irá para produção passou na suíte de teste pré-estabelecida. Após a implantação é executado um teste de fumaça no ambiente de produção.

Figura 21 – Processo da Entrega de Versão



Fonte: Elaborado pelo autor (2022)

6.2.4.7 Automatizar o Processo de Compilação

A automatização o processo de compilação, se deu de três formas diferentes, variando conforme o ambiente. No ambiente de desenvolvimento, a automatização se deu através do *Makefile*, o qual configura todo ambiente, desde que o Python já esteja instalado.

No ambiente de testes, durante a execução do pipeline, a automatização se deu através do arquivo *Dockerfile*, no qual foi usado para configurar o contêiner onde as etapas do pipeline com o Jenkins eram executadas.

No ambiente de produção, a compilação é executada pelo script de implantação construído pelo Fabric, detalhado na seção seguinte.

6.2.4.8 Automatizar o Processo de Implantação

O processo de implantação nos ambientes de homologação e produção, foi desenvolvido um *script* Python com a biblioteca Fabric. Com o Fabric é possível executar comandos shell remotamente por SSH.

No *script* de *deploy* do projeto piloto, contém um dicionário com o nome dos ambientes

e os IPs dos servidores. O script de deploy se conecta via SSH no primeiro IP da lista de IPs do ambiente informado, e executa as seguintes etapas:

1. Para o servidor de aplicação;
2. Atualiza o código do fonte a aplicação usando o Git;
3. Instala as dependências da aplicação;
4. Aplica as migrações no banco de dados;
5. Coleta e envia os arquivos estáticos para o Rede de Entrega de Conteúdo, em inglês *Content Delivery Network* (CDN);
6. Cria pasta de *upload*;
7. Atualiza as informações de implantação;
8. Atualiza as permissões da pasta da aplicação;
9. Atualiza as configurações do servidor de aplicação e do servidor web;
10. Inicia o servidor de aplicação e reinicia o servidor web; e
11. Verifica se os servidores iniciaram corretamente.

Após executar todas as etapas acima, o script se desconecta do servidor e se conecta no próximo IP da lista. O script executará as etapas acima para todos o IPs da lista e ao final exibirá um relatório de quanto tempo durou a execução das etapas em cada servidor e a duração total da execução em todos os servidores.

6.2.4.9 Automatizar Testes Estruturais

Os testes unitários e de integração foram construídos usando o módulo de testes do *framework* Django que tem como base a biblioteca padrão Python *unittest*. Durante a execução dos testes, o Django cria um banco de dados de teste e inicia um servidor de aplicação de teste, com o qual se é possível testar as respostas das requisições web. Ao final da execução dos testes o banco é excluído.

6.2.4.10 Automatizar Testes Funcionais

Os testes funcionais foram escritos em linguagem natural usando a biblioteca *behave-django*. Esses testes cobriam o fluxo principal das funcionalidades que não estavam cobertas pelos testes de integração. Essa abordagem, reduz o retrabalho de testar a mesma funcionalidade mais de uma vez, além de reduzir o tempo de execução dos testes de aceitação.

6.2.4.11 Configurar Pipeline

A configuração do pipeline de integração e entrega contínua foi realizada inicialmente através da criação de um pipeline declarativo usando o *Jenkinsfile* (disponível por completo no Apêndice F), como descrito na subseção 6.2.3.1. No entanto, por causa da incompatibilidade do Gitea com o Jenkins e pela falta de ferramentas do Gitea que ajudassem a gerenciar o repositório, o Gitea foi substituído pelo GitLab.

A integração do GitLab com o Jenkins se deu normalmente, no entanto, com o passar do tempo, um novo problema de incompatibilidade surgiu. Um dos problemas que surgiu, era a impossibilidade de reexecutar o pipeline, como quando acontecesse algum problema de rede, por exemplo, sendo necessário um novo commit, mesmo sem o código ter sofrido nenhuma alteração.

Por fim, a equipe decidiu usar o próprio servidor de integração do GitLab, o GitLab CI. O pipeline foi reescrito e encontra-se por completo no Apêndice G.

6.2.4.12 Avaliar Uso do CI/CD

Coletar métricas nos ajudam a identificar gargalos e as melhorias proporcionadas pela nova prática. Para avaliar o uso do CI/CD, pode-se coletar relacionadas ao processo de entrega, a qualidade do código e do produto, conforme descrito na seção 6.4.

6.2.5 Entrevistas com a Equipe

As entrevistas foram realizadas com 4 desenvolvedores de forma pessoal, visando captar as impressões deles com relação as práticas de CI/CD adotadas no IFAC.

Dois conjuntos de questões foram investigados neste estudo. O primeiro conjunto tratava

do conhecimento dos desenvolvedores com relação as práticas de CI/CD. O segundo conjunto avaliou a adoção do pipeline, especificamente as percepções do participantes sobre o uso das práticas de CI/CD.

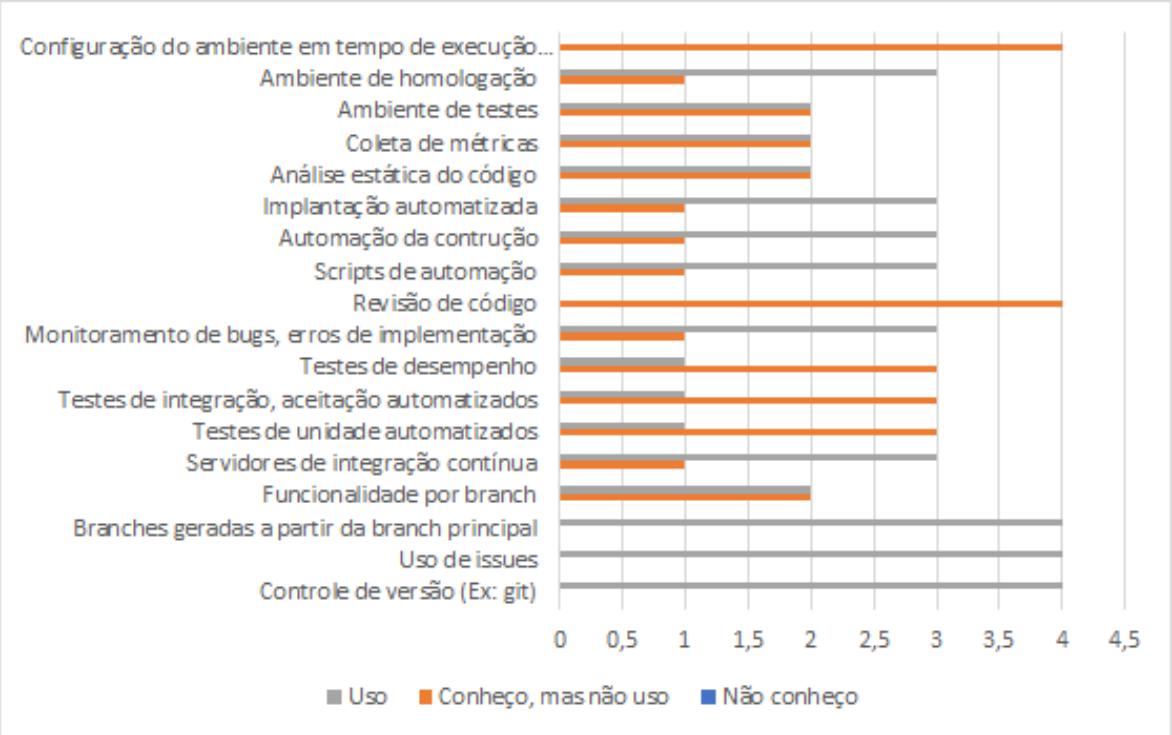
6.2.5.1 Conhecimento sobre as práticas

O conhecimento sobre as práticas contínuas ajudam a facilitar o processo de adoção e utilização no dia a dia.

Para se avaliar o conhecimento da equipe sobre as práticas de CI/CD utilizadas no IFAC, a equipe dos dois projetos sobre tal conhecimento.

Ao serem analisadas as respostas, observa-se que todos os desenvolvedores informaram que conheciam, porém nem todas eram utilizadas. Em novos projetos, somente a integração contínua é utilizada, e em projetos legados, não é usado nem o CI, ou seja, poucos são os projetos que utilizam CI/CD. A Figura 22 ilustra o conhecimento da equipe sobre as práticas de CI/CD.

Figura 22 – Gráfico sobre o conhecimento da equipe das práticas de CI/CD

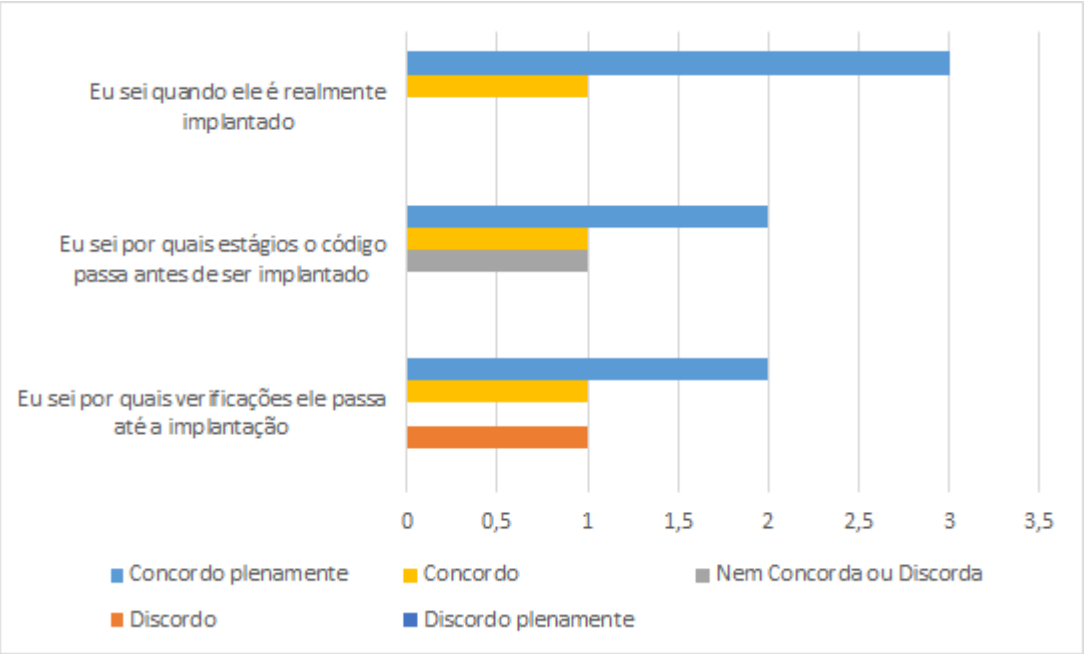


Fonte: Elaborado pelo autor (2022)

Os participantes foram questionados sobre a transparência do pipeline CI/CD, nota-se que todos os desenvolvedores tem conhecimento do que é executado no pipeline, seus estágios,

verificações e quando o código é implantado. A Figura 23, ilustra a percepção dos desenvolvedores com relação a transparência do pipeline.

Figura 23 – Gráfico sobre a percepção das etapas do pipeline



Fonte: Elaborado pelo autor (2022)

6.2.5.2 Impressões dos desenvolvedores

Esse conjunto de questões buscou avaliar a adoção do pipeline, sob a visão da equipe de desenvolvimento, buscando entender e coletar as percepções da equipe sobre o uso do CI/CD.

Quando interrogados sobre a percepção em relação a entrega contínua e se achavam que ela contribuía positivamente ao projeto, nota-se nos relatos que todos os participantes reconhecem os benefícios do CI/CD. Os benefícios relatados foram, a automação e a execução dos testes automatizados, o que faz com que sejam descobertos menos erros em produção e durante a implantação:

> **Desenvolvedor 1** - *"Sim. Sempre corrigindo e/ou melhorando desempenho, funcionalidades ou mudanças e/ou adaptações nas regras de negócio."*

> **Desenvolvedor 2** - *"Acredito que a automação reduz erros."*

> **Desenvolvedor 3** - *"Sim, porque com a integração e a entrega contínua o código é testado e implantado no ambiente de homologação antes de ir para produção. Assim temos a certeza que o novo código não introduz um erro/bug."*

> **Desenvolvedor 4** - *"Entrega contínua representa um grande avanço para a organização, uma vez que padroniza todo o processo e formaliza tarefas que antes eram feitas de informalmente."*

Quando questionados sobre se existia algum impedimento para realizar a entrega contínua, observa-se que nos relatos dos participantes que, os empecilhos são a configuração do pipeline e a falta de comunicação. Apresentando uma relação com os desafios D2. Curva de aprendizado íngreme e D5. Falta de comunicação, encontrados na RSL.

Um dos participantes relatou que o problema da configuração está em não ter a possibilidade de testar. Já outro desenvolvedor, levantou um problema de comunicação com o cliente, dificultando na validação da funcionalidade. Segue abaixo as transcrições das respostas:

> **Desenvolvedor 1** - *"Geralmente por conta da equipe ou responsável por definir, fiscalizar, usar e atestar as regras de negócio."*

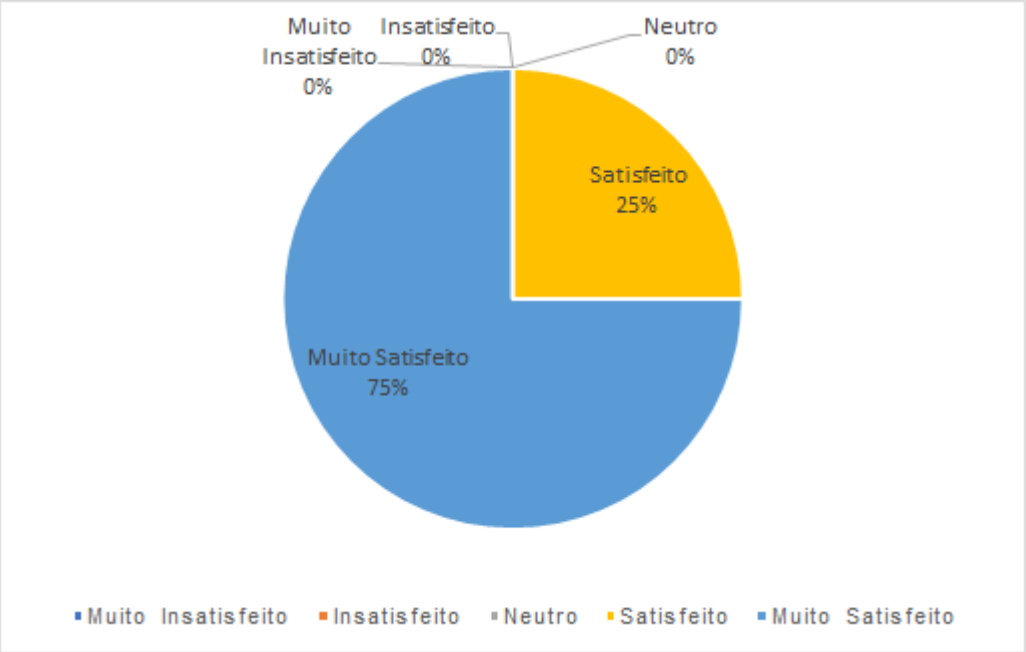
> **Desenvolvedor 2** - *"A complexidade da configuração é um empecilho."*

> **Desenvolvedor 3** - *"O maior problema é a configuração do pipeline, pois não há uma forma de testar a não ser realizando commits, o que acaba gerando diversos commits de ajustes do arquivo de configuração do pipeline."*

> **Desenvolvedor 4** - *"Entendo que, no cenário atual, não."*

Por fim, foi questionado como eles avaliavam a adoção da integração e entrega contínua, em uma escala de 1 a 5, sendo que 1 muito insatisfeito e 5 muito satisfeito, como pode ser visto na Figura 24. Somente um desenvolvedor avaliou como satisfeito e quando questionado do porque ele ressaltou a dificuldade de configurar o pipeline pela primeira vez ou adicionar uma etapa nova.

Figura 24 – Gráfico sobre a satisfação da equipe



Fonte: Elaborado pelo autor (2022)

6.2.6 Lições Aprendidas

Durante o uso e adoção de CI/CD sugiram alguns desafios, sendo que alguns desses foram identificados e listados no capítulo anterior. Seguem os desafios vivenciados, contextualizando o problema enfrentado:

- D2. Curva de aprendizado íngreme: A dificuldade para aprender e entender como o pipeline funciona nas diferentes ferramentas, tornar o processo da configuração inicial demorado. O mesmo problema pode acontecer caso a equipe queira adicionar mais um estágio no pipeline.
- D8. Falta de Habilidades: A equipe de desenvolvimento embora tivesse experiência em desenvolver sistemas, não tinha experiência com construção de testes automatizados;
- D18. Arquitetura Altamente Acoplada: O projeto piloto desenvolveu uma aplicação monolítica, o que dificultou a criação dos testes automatizados, se fazendo necessário a inserção de uma grande quantidade de dados iniciais, além das dependências entre os modelos;
- D19. Dependências: As bibliotecas utilizadas dependiam de pacotes que precisavam serem instalados antes de instalá-las;

- D21. Falta de Estratégia de Testes: A falta de uma estratégia de testes dificultou a execução e criação dos testes automatizados, pois tinham dificuldades em saber o que testar e qual ferramenta usar;
- D22. Falta de Testes: A falta de testes resultou em diversas entregas de correções de erros;
- D23. Baixa Cobertura de Testes: A equipe observou que precisava melhorar a cultura de testes automatizados, visto que sua adoção no projeto piloto foi tardia o que resultou em uma cobertura de testes relativamente baixa;
- D30. Configuração Manual de Software: A falta da automação da configuração e provisionamento dos servidores gerou atrasos na primeira implantação em cada ambiente, fazendo com que a equipe leva-se 1 (um) dia para concluir as configurações;
- D31. Falta de Recursos: A falta de recursos financeiros impossibilitou a contratação de treinamentos e a adoção de uma ferramenta de gerenciamento de projetos que pudesse ser integrada com o Gitea, Jenkins e SonarQube;
- D32. Esforço Inicial: Um esforço inicial foi necessário para integrar e configurar as ferramentas escolhidas, que gerou um atraso na criação e configuração do pipeline;
- Burocracia: Aliada a falta de recursos, a burocracia de um processo administrativo para contratar um treinamento e/ou uma ferramenta, atrasaria a pesquisa visto que teríamos que esperar a finalização do mesmo;
- A falta de um único *script* de configuração da aplicação, ocasionou correções na etapa de instalação das dependências no *Jenkinsfile*;
- Demora na execução dos testes: Inicialmente a execução dos testes demorava cerca de 6 (seis) minutos, devido a dependências desnecessárias para serem utilizadas em um ambiente de testes;
- Integração entre as ferramentas: A integração do Jenkins com o SonarQube, teve alguns problemas para encontrar o *sonar-scanner* instalado no servidor do Jenkins. Essa dificuldade deve-se devido a utilização de um contêiner para a construção e execução dos testes no pipeline. Outra dificuldade enfrentada foi na integração do SonarQube com o Gitea;

Algumas lições aprendidas durante a implantação de CI/CD são listadas abaixo, como forma de ajudar outros pesquisadores e desenvolvedores a adotar tais práticas:

- A interferência de atividades externas prejudica a produtividade da equipe, uma vez que essa tem que dividir sua atenção;
- Designar um responsável pela implantação de CI/CD de fora do projeto, uma vez que no projeto piloto um dos desenvolvedores foi o responsável pela configuração do pipeline, e portanto, esse teve que dividir sua atenção entre a configuração e desenvolvimento do projeto;
- Adotar um sistema para o gerenciamento de projetos/tarefas, uma vez que usar o *issue tracker* nativo do Gitea prejudicou a visualização do andamento das tarefas pela equipe do projeto, visto que o mesmo não permite a criação de quadros de tarefa Kanban, dificultando a visibilidade do projeto;
- O planejamento das etapas do pipeline deve ser realizado após o mapeamento e definição do fluxo de valor;
- Utilizar um contêiner Docker para executar os testes automatizados;
- Configurar o servidor CI para verificar o repositório de código através do *webhook*, ao invés do agendamento;
- Executar os testes unitários em paralelo, embora sua execução seja mais rápida, dificulta o *feedback* dos desenvolvedores, uma vez que, no caso do Django, não identifica o teste que não passou, atrasando a descoberta do erro;
- Implantar em produção versões específicas do código, através de tags ou ramificações de versões o que gera rastreabilidade, ao invés de implantar a ramificação principal diretamente.

6.3 PROJETO IRMÃO: O PROJETO CACHALOTE

Dentre os projetos na fila de construção, escolheu-se este projeto devido ao seu escopo e tamanho serem parecidos com o projeto Manhanah, além de sua importância no contexto acadêmico.

Assim como o projeto Manhanah, o projeto Cachalote foi desenvolvido utilizando metodologias ágeis, no entanto, o processo utilizado era manual, com seu código sendo enviado para servidor através do SFTP, conforme descrito na subseção 6.1.2.2 e na subseção 6.1.2.3 respectivamente.

O início do projeto Cachalote se deu em março de 2018, oriundo da iniciativa da COSIS para melhorar a gestão dos eventos, visto que na época, para cada evento era criado um sistema. Portanto, o projeto Cachalote surgiu com o intuito de que fosse criado uma plataforma para todos os tipos de eventos acadêmicos, tendo foco no gerenciamento do evento, inscrições e submissões de trabalhos, sendo lançada em outubro deste mesmo ano.

Com a alta demanda de eventos, os gestores estavam encontrando dificuldades para configurá-los e gerenciá-los reclamando de questões relacionadas a usabilidade e o alto número de erros em produção. Devido a dificuldade de manutenção por parte dos desenvolvedores ocasionados por fluxos complexos, a COSIS, resolveu criar uma nova versão do sistema.

O projeto Cachalote, foi então reaberto, para que o sistema fosse reconstruído. A versão 2.0 como foi chamado pela equipe foi desenvolvida tendo como foco principal a usabilidade por parte dos gestores de eventos e a simplificação dos fluxos existentes.

Para a execução do projeto foi definida a equipe composta por 1 (um) desenvolvedor e o coordenador da COSIS. O Quadro 13 apresenta o perfil dos colaboradores do projeto Cachalote.

Quadro 13 – Perfil dos colaboradores do projeto Cachalote

Colaborador	Faixa Etária	Tempo de Instituição (anos)	Experiência com desenvolvimento e implantação (anos)	Cargo
Coordenador	30-34 anos	5	16	Analista de TI
Desenvolvedor 1	35-39 anos	6	16	Analista de TI

Fonte: Elaborado pelo autor (2022)

6.3.1 Descrição do projeto

O projeto Cachalote tem como objetivo principal fornecer uma ferramenta para cadastro e gerenciamento de eventos realizados pelo IFAC. Dispõe de módulos de gerenciamento do evento, inscrição on-line, submissão de trabalhos científicos e/ou acadêmicos, acompanha-

mento das submissões, avaliações dos trabalhos, controle de presença e emissão de certificados.

6.3.2 Preparação para o projeto

Com base no estudo preliminar e nos processos de desenvolvimento e entrega de software, foram sugeridas técnicas para que este projeto possa servir como fonte de comparação da adoção das práticas de integração e entrega contínua.

- Tendo em vista a cultura da equipe de não coletar métricas, foi orientado como registrar os dados com a finalidade de que possam ser coletadas métricas relacionadas ao ciclo de entrega do produto;
- Foi configurada a ferramenta SonarQube com o intuito de coletar métricas relacionadas a qualidade do código.

6.4 ANÁLISE E RESULTADOS

Esta seção envolve a análise dos dados coletados após adoção da integração e entrega contínua. O resultado da análise é um relatório sobre o impacto da adoção dessas práticas no IFAC, relacionados ao processo de entrega e qualidade do código e do produto e qualquer outra observação relevante sobre o tema. Além disso, o relatório é baseado em uma análise qualitativa no que diz respeito às informações obtidas na diretoria sobre suas práticas de desenvolvimento e entrega de software.

Para realizar essa análise, o *Goal Question Metric* (GQM)(BASILI; CALDIERA; ROMBACH, 1994) foi usado para definir o objetivo do estudo e derivar as questões que deveriam ser respondidas a fim de determinar se o objetivo foi alcançado ou não. Além disso, o objetivo é uma declaração do que o patrocinador espera alcançar e quais são as bases das hipóteses do estudo (KITCHENHAM; PICKARD, 1998; RUNESON; HÖST, 2008). Assim, definimos o objetivo da seguinte forma:

G. Analisar a adoção de integração e entrega contínua para efeito de avaliação quanto à viabilidade do ponto de vista do pesquisador no contexto do Instituto Federal do Acre.

Depois de definir o objetivo, ele é então refinado em questões que tentam caracterizar os objetos de medição com relação a um aspecto de qualidade selecionado e determinar sua

qualidade do ponto de vista selecionado (WOHLIN et al., 2012). As questões são apresentadas da seguinte forma:

Questão 1: Quão apropriada é a adoção de CI/CD para o processo de entrega de software?

Nesta questão, tentamos identificar a relação custo-benefício da adoção de integração e entrega contínua se comparada as práticas atuais de entrega de software.

Questão 2: Qual o impacto da adoção de CI/CD na qualidade do código gerado?

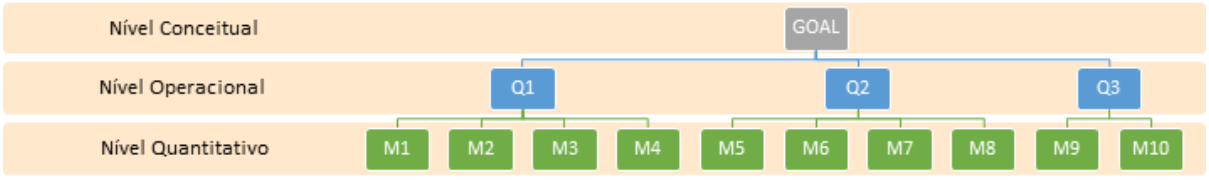
Nesta questão, tentamos identificar a relação da adoção de CI/CD na qualidade do código produzido.

Questão 3: Qual o impacto da adoção de CI/CD na qualidade do produto de software?

Nesta questão, tentamos identificar a relação da adoção de CI/CD na qualidade do produto de software.

Em seguida, um conjunto de métricas é associado a cada questão para respondê-la de forma quantitativa, seja de forma objetiva ou subjetiva, conforme afirma Wohlin et al. (2012). Para orientar a interpretação dos dados, foi definido um *framework* de medição com três níveis, conforme ilustrado pela estrutura hierárquica na Figura 25.

Figura 25 – Framework de Medição



Fonte: Elaborado pelo autor (2022)

As métricas foram objetivamente definidas da seguinte forma:

O objetivo principal deste estudo é avaliar a adoção de integração e entrega contínua quanto à viabilidade. A viabilidade, neste caso, é definida pela Qualidade e pela Velocidade, no IFAC.

O processo de entrega pode ser avaliado quanto a velocidade por meio da análise qualitativa de quatro medidas, sendo elas o *lead time*, *cycle time*, frequência de implantação e tempo de implantação (ver Tabela 6). Entendemos como um processo de entrega eficiente e viável o aumento da frequência de implantação e a diminuição das outras métricas.

A Figura 26 apresenta a relação do *lead time* com o *cycle time*, visto que o que diferencia ambas as métricas é o evento de início, uma vez que o tempo das duas métricas termina quando a funcionalidade é implantada em produção. O *lead time* começa a ser contado desde

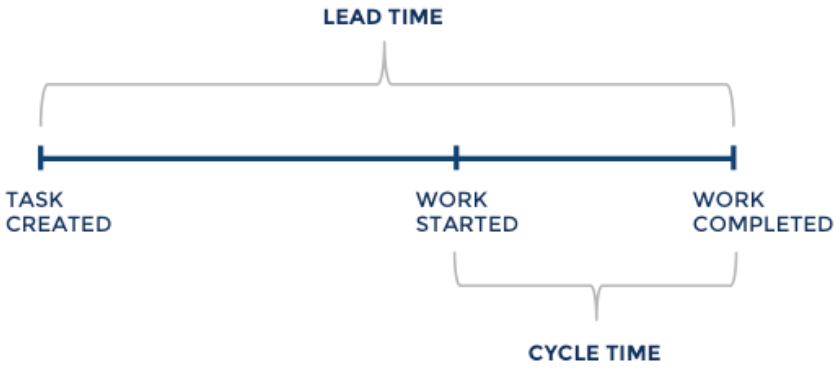
Tabela 6 – Métricas usadas para analisar o Processo de Entrega

Métrica	Descrição
M1. *Lead Time*	Tempo que uma funcionalidade leva desde a sua solicitação até ser entregue ao cliente
M2. *Cycle Time*	Tempo que uma funcionalidade é considerada "em progresso" até o momento em que ela entregue para o usuário final
M3. Frequência de Implantação	Frequência de entrega do código para o usuário final ou para o ambiente de produção
M4. Tempo de Implantação	Tempo desde o pedido de uma implantação até a implantação em produção

Fonte: Elaborado pelo autor (2022)

a solicitação da funcionalidade, já o *cycle time* tem seu início como o momento que a equipe começa a trabalhar na funcionalidade.

Figura 26 – Relação do *Lead Time* com o *Cycle Time*



Fonte: Blog da Screenful (2015)[22]

A qualidade do código pode ser avaliada por meio da análise qualitativa de quatro medidas, sendo elas o número de vulnerabilidades, o número de *bugs*, o número de *code smells* e o número de complexidades (ver Tabela 7). Entendemos como aumento da qualidade do código a diminuição dessas métricas.

A qualidade do produto pode ser avaliada por meio da análise qualitativa de duas medidas, são elas o número de defeitos e a cobertura de testes automatizados (ver Tabela 8). Entendemos como aumento da qualidade do produto a diminuição dessas métricas.

Para evitar vieses e garantir a validade interna, é necessário criar uma base sólida para avaliar os resultados do estudo (WOHLIN et al., 2012). Portanto, foi escolhido o método de

[22] Disponível em: <https://screenful.com/blog/software-development-metrics-cycle-time>. Acesso em: 02 dez. 2020

Tabela 7 – Métricas usadas para analisar a Qualidade do Código

Métrica	Descrição
M5. Número de Vulnerabilidades	O número de vulnerabilidades de segurança
M6. Número de *bugs*	O número de *bugs* que representam algo de errado no código
M7. Número de *code smells*	O número de problemas de manutenção
M8. Número de complexidades	O número de caminhos através do código

Fonte: Elaborado pelo autor (2022)

Tabela 8 – Métricas usadas para analisar a Qualidade do Produto

Métrica	Descrição
M9. Número de defeitos	O número de defeitos reportados
M10. Cobertura de Testes Automatizado	Porcentagem de testes automatizados

Fonte: Elaborado pelo autor (2022)

comparação do projeto irmão que serviu como linha de base. Este método, envolve dois projetos, o projeto piloto que utiliza o novo método e o projeto irmão, o atual (KITCHENHAM; PICKARD; PFLEEGER, 1995).

A Tabela 9 apresenta os dados de cada projeto. A primeira coluna se refere ao nome do projeto. As colunas restantes apresentam: domínio do sistema, número de linhas de código, número de linhas comentadas e o número de arquivos. Todos os dados foram coletados usando a ferramenta SonarQube. Todos os sistemas são WEB e foram desenvolvidos utilizando a linguagem Python com o *framework* web Django.

Tabela 9 – Dados gerais dos sistemas

Projeto	Domínio	# LOC	# Linhas Comentadas	# Arquivos
Projeto Manhanah	Gerenciamento Eletrônico de Documentos	20553	602	203
Projeto Cachalote	Sistema Gestor de Eventos	25597	495	303

Fonte: Elaborado pelo autor (2022)

6.4.1 Questão 1: Quão apropriada é a adoção de CI/CD para o processo de entrega de software?

Para análise das métricas *lead time* e *cycle time* foram coletadas e analisadas 10 versões de cada projeto.

O Quadro 14 demonstra as estatísticas descritivas computadas (descritas na seção 6.4), onde a primeira linha demonstra os dados das versões do projeto irmão utilizando o processo de entrega atual e a segunda linha demonstra os dados das versões após a adoção das práticas de integração e de entrega contínua.

Quadro 14 – Estatísticas descritivas Métricas de Velocidade

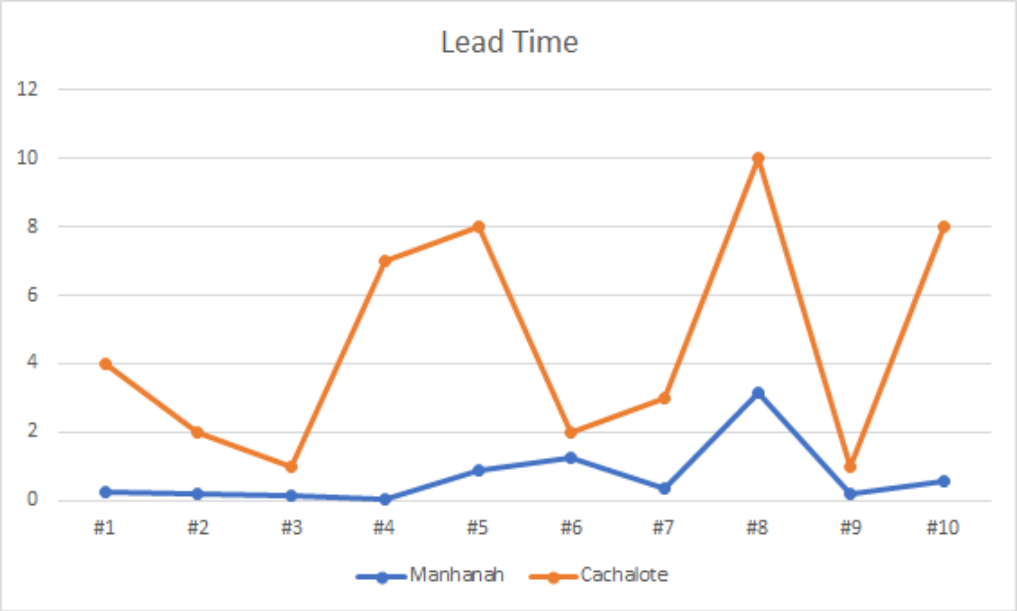
Métricas	Mínimo	Mediana	Máximo	Desvio Padrão	Média
M1. *Lead time* (Projeto Irmão)	1 dia	3,5	10 dias	3 dias e 8h09min	4,6 dias
M1. *Lead time* (Projeto Piloto)	1h14min	11h12min	3 dias e 19h52min	1 dia e 8h04min	18h43min
M2. *Cycle time* (Projeto Irmão)	1 dia	3 dias	10 dias	2 dias e 23h35min	3,7 dias
M2. *Cycle time* (Projeto Piloto)	17min	2h13min	2 dias	14h23min	7h12min
M3. Frequência de Implantação (Projeto Irmão)	0	1	3	0,782	1,111
M3. Frequência de Implantação (Projeto Piloto)	0	2	9	2,99	3,22
M4. Tempo de Implantação (Projeto Irmão)	3min21s	4min16s	4h10min	1h17min	29min
M4. Tempo de Implantação (Projeto Piloto)	8min	8min23s	8min40s	14s	8min21s

Fonte: Elaborado pelo autor (2022)

M1. *Lead time*. Para calcular esta métrica, foi registrado quando a a funcionalidade foi solicitada e quando foi disponibilizada em produção.

A Figura 27 apresenta um gráfico com os dados coletados referentes ao *lead time* dos dois projetos. O eixo-X representa as versões analisadas, enquanto o eixo-Y consiste na quantidade de dias referente ao tempo para disponibilizá-la em produção. Observa-se que com a utilização das práticas de CI/CD o *lead time* ficou menor comparado a prática de entrega atual. Verificamos que na primeira versão analisada do projeto piloto, o tempo elevado para a entrega da versão deu-se por causa da espera que a funcionalidade teve para que a equipe pudesse começar a trabalhar nela e na décima versão o atraso da entrega deu-se devido ao tempo de implementação. Conforme apresentado no Quadro 14 o número máximo do *lead time* foi menor com a utilização das práticas de CI/CD.

Figura 27 – Gráfico Comparativo do *Lead Time*



Fonte: Elaborado pelo autor (2022)

M2. *Cycle time*. Para calcular esta métrica, foi registrado o início do trabalho da equipe de desenvolvimento para implementar a funcionalidade.

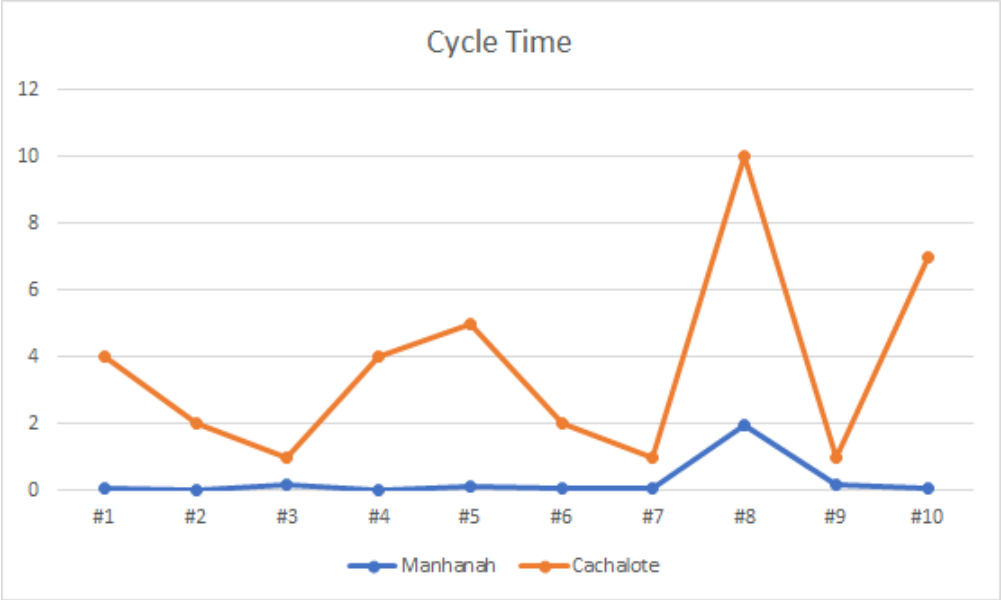
A Figura 28 apresenta um gráfico com os dados coletados e traz um comparativo do *cycle time* dos dois projetos. O eixo-X representa as versões analisadas, enquanto o eixo-Y consiste na quantidade de dias referentes ao tempo desde o início do trabalho na funcionalidade até sua disponibilização em produção. Observa-se que com a utilização das práticas de CI/CD o *cycle time* ficou menor comparado a prática de entrega atual. Ao verificarmos a primeira versão do projeto piloto seu *cycle time* foi de 5h55min, embora tenha apresentado um *lead time* elevado foi devido a espera, conforme mencionado anteriormente. O principal resultado observado é o baixo *cycle time* com a adoção das práticas CI/CD.

M3. Frequência de Implantação. O número de implantações foi calculado contabilizando a quantidade de entregas em produção juntamente com o tipo de funcionalidade. Para análise e coleta dessa métrica foram consideradas 9 (nove) semanas.

A coleta da frequência de implantação do projeto irmão deu-se durante os meses de outubro e novembro de 2020, pois tratava-se do início da fase de sustentação do sistema e de um período de uso intensivo do sistema devido a realização da 5ª (quinta) edição do Congresso de Ciência e Tecnologia do IFAC (CONC&T), que consiste no maior evento anual de divulgação científica organizado pelo IFAC.

A coleta da frequência de implantação do projeto piloto deu-se durante os meses de no-

Figura 28 – Gráfico Comparativo do *Cycle Time*



Fonte: Elaborado pelo autor (2022)

vembro e dezembro de 2020, pois tratava-se de um período em que o sistema obteve melhorias visando a alta demanda devido a entrega do PIT do segundo semestre e do RIT do primeiro semestre de 2020, no início do segundo semestre de 2020, que devido a pandemia do COVID-19, teve seu o início adiado para janeiro de 2021.

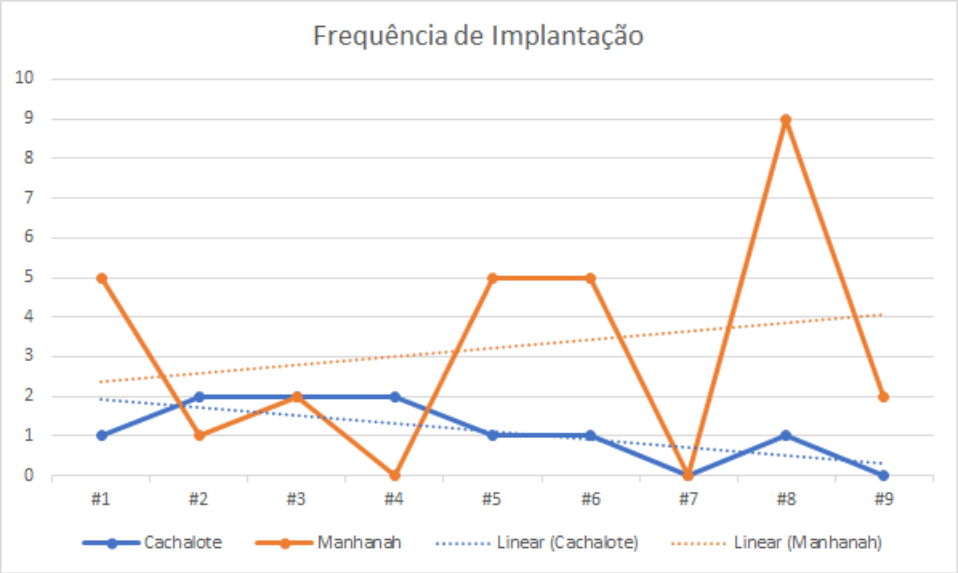
A Figura 29 apresenta o gráfico comparativo da frequência de implantação. O eixo-X representa as quantidades de entregas por semana, enquanto o eixo-Y consiste no número de semanas. Durante esse período foram realizadas 10 (dez) entregas de versões do projeto irmão, sendo que 4 (quatro) dessas entregas foram melhorias e 6 (seis) correções de erro. No projeto piloto, foram realizadas 29 (vinte e nove) entregas, sendo 5 (cinco) melhorias e 24 (vinte e quatro) correções de erro. Pode-se observar que o projeto piloto possui uma frequência de implantação em alta, no entanto, o alto índice de correções de erro deve-se a *bugs* no layout.

M4. Tempo de Implantação. Para calcular esta métrica, foi registrado a data do pedido de implantação e a data da implantação.

O processo de implantação (ver subseção 6.2.3) do projeto irmão diferencia-se do projeto piloto devido a equipe não identificar as versões implantadas. Portanto, para este projeto foi registrado a data e hora na qual um membro da equipe acessou o servidor para realizar as etapas de implantação. Já para o projeto piloto, o *commit* identificando a versão foi considerado para registro da data e hora inicial.

No projeto piloto o tempo de implantação é mais constante, havendo pequenas variações

Figura 29 – Gráfico Comparativo da Frequência de Implantação



Fonte: Elaborado pelo autor (2022)

de alguns segundos entre uma implantação e outra, ocasionado pela latência de rede, sendo que em 10 (dez) implantações o menor tempo registrado foi de 8min e o maior de 8min40s.

No projeto irmão o tempo de implantação é instável, pois para as mesmas 10 (dez) implantações o menor tempo registrado foi de 3min e o maior foi de 4h. A implantação que demorou 4h, foi devido a um erro que ocorreu, no qual foi necessário reconfigurar o projeto no ambiente de produção, ficando o sistema inativo pelo mesmo tempo de implantação. As demais implantações, ficaram variando entre 3min e 6min, dependo se a alteração necessitava republicar os arquivos estáticos, via SFTP.

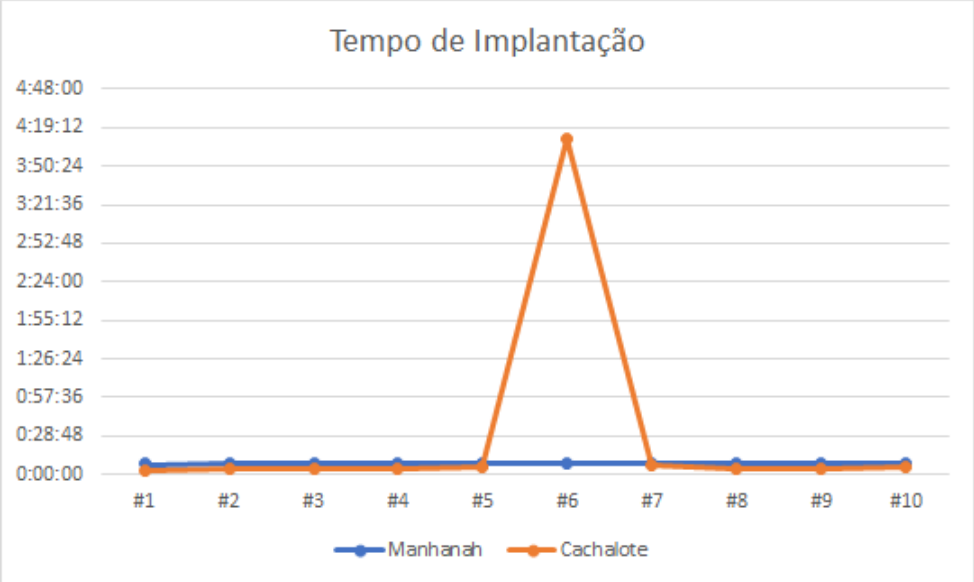
A Figura 30 apresenta um gráfico comparativo do tempo de implantação dos dois projetos. O eixo-X representa as entregas analisadas, enquanto o eixo-Y consiste no tempo que levou cada processo de implantação. Ao observarmos a diferença de tempo de implantação do estudo preliminar para o período da coleta das métricas, observamos que o método de envio do código para os servidores foi modificado de SFTP para a atualização do código via Git, diminuindo o tempo de implantação.

Embora o tempo de implantação do projeto irmão seja menor, não há uma rastreabilidade das versões entregues, nem um processo definido para implantar, ou seja, as versões não são identificadas nem são criadas tags, sendo executados somente os passos para implantar uma aplicação Django, além disso, os desenvolvedores escolhem não executar alguns passos, como, por exemplo, a coleta dos arquivos estáticos. Durante a implantação #6 (número seis) do

projeto irmão, ocorreu um erro que fez com que a equipe tivesse que reconfigurar o ambiente devido a alterações realizadas diretamente em produção e posteriormente adicionada no Git, causando um conflito durante atualização do código (*git pull*).

Já no projeto piloto, por ser definido que todas as modificações deveriam ser commitadas no Git possibilitando a execução do pipeline CI/CD para posteriormente serem disponibilizadas em produção, o erro que aconteceu com a implantação #6 (número seis) do projeto irmão não irá acontecer. Além disso, a atualização em produção é automatizada e só acontece quando é criada uma tag e, só então, o código em produção é atualizado conforme essa tag (*git checkout* na tag).

Figura 30 – Gráfico Comparativo do Tempo de Implantação



Fonte: Elaborado pelo autor (2022)

6.4.2 Questão 2: Qual o impacto da adoção de CI/CD na qualidade do código gerado?

A coleta das métricas de qualidade do código se deu através da ferramenta SonarQube. No entanto, a ferramenta retorna falsos positivos relacionados a *bugs* e *code smells*, portanto, foi realizada uma reclassificação dos dados coletados.

M5. Vulnerabilidade. O número de Vulnerabilidades foi calculado com base no número de vulnerabilidades de segurança encontrados no código.

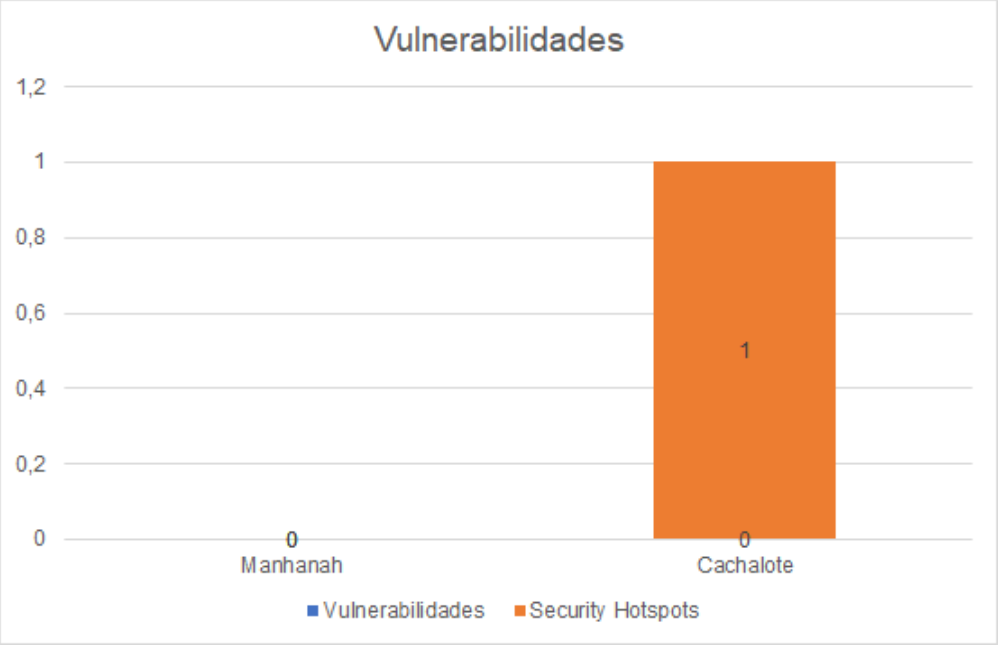
A Figura 31 apresenta um gráfico comparativo com os dados coletados referentes ao nú-

mero de vulnerabilidades. Embora em nenhum dos dois projetos tenham sido encontradas vulnerabilidades no código, no projeto Cachalote, o SonarQube encontrou um *security hotspots*, ou seja, uma parte do código sensível à segurança que o desenvolvedor precisa revisar.

Devido ao número baixo de alertas de vulnerabilidades e de *security hotspots*, foi levantado o histórico de análises do projeto Manhanah e Cachalote e verificou-se que não houve alertas de vulnerabilidades em nenhum dos dois projetos. Com relação ao *security hotspots*, foram constatados vários alertas, classificados como falsos positivos, nos dois projetos.

Portanto, a experiência dos desenvolvedores pode ter impactado nessa análise, por estarem seguindo as práticas relacionadas a segurança no desenvolvimento de sistemas.

Figura 31 – Gráfico Comparativo de Vulnerabilidades



Fonte: Elaborado pelo autor (2022)

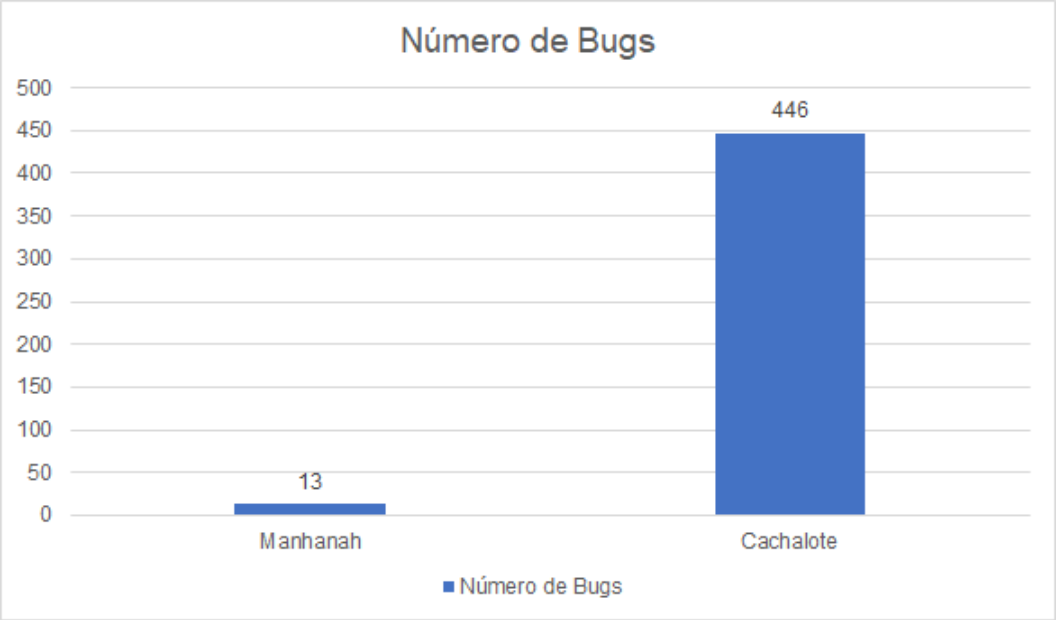
M6. *Bug*. O número de *Bugs* foi calculado com base no número de erros encontrados no código, que estão ligados a confiabilidade do código.

A Figura 32 apresenta um gráfico comparativo entre os projetos com dados coletados referentes ao número de *bugs*. Como podemos observar com a adoção do CI/CD o número de *bugs* do projeto é menor.

M7. *Code Smells*. O número de Code Smells foi calculado com base no número de problemas para manutenções posteriores no código.

A Figura 33 apresenta um gráfico comparativo com os dados coletados referentes ao número de *code smells*. Como pode-se observar o projeto Manhanah possui mais *code smells*

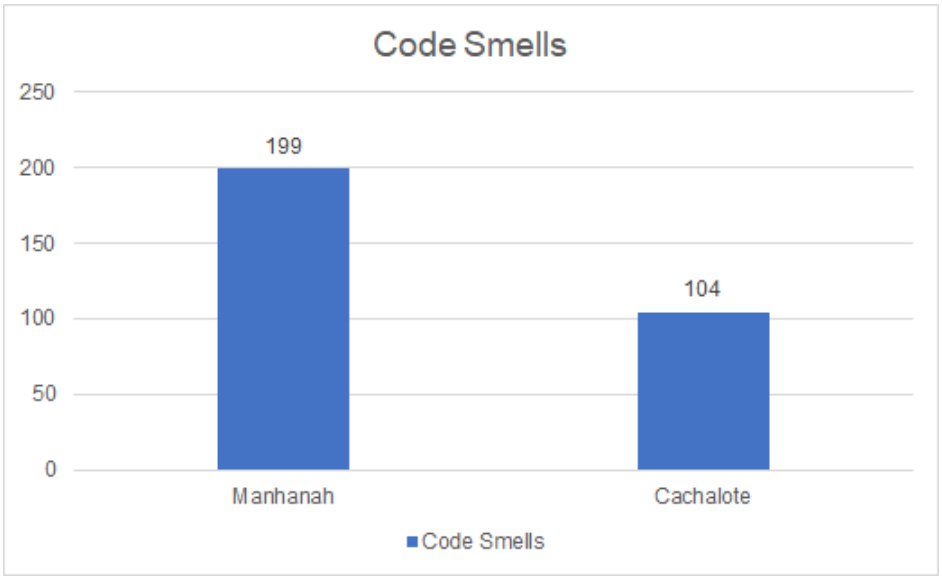
Figura 32 – Gráfico Comparativo do Número de *Bugs*



Fonte: Elaborado pelo autor (2022)

que o projeto Cachalote. No entanto, essa quantidade elevada de *code smells* poderia ser menor, caso a adoção da ferramenta de análise estática de código no pipeline tivesse se dado no começo do desenvolvimento.

Figura 33 – Gráfico Comparativo de *Code Smells*

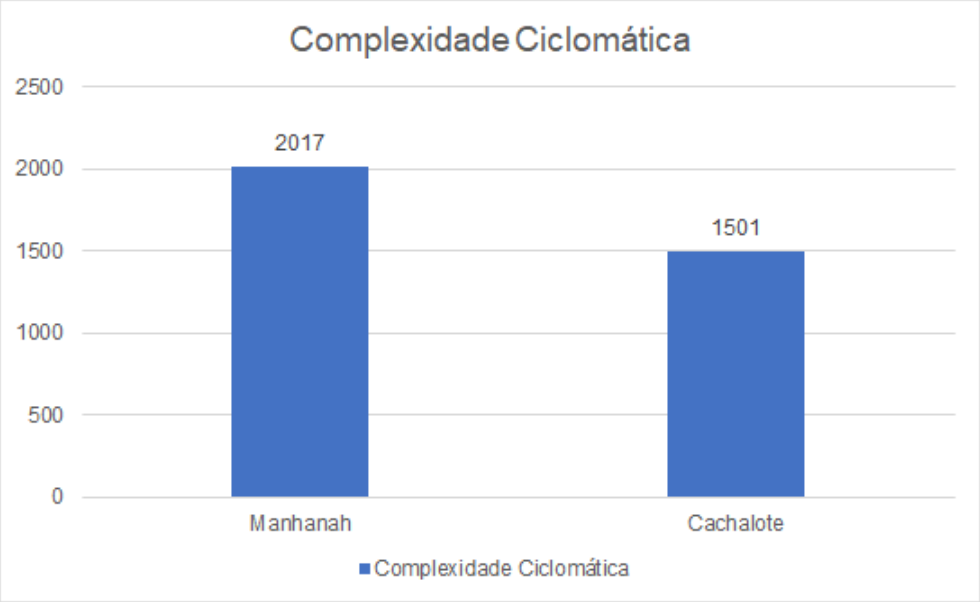


Fonte: Elaborado pelo autor (2022)

M8. Complexidade Ciclomática. O número de Complexidades foi calculado com base no número de caminhos através do código. Sempre que o fluxo de uma função se divide, o contador de complexidade é incrementado em 1.

A Figura 34 apresenta um gráfico comparativo com os dados coletados referentes a complexidade ciclomática de cada projeto. Como pode ser observado a complexidade do projeto Manhanah é maior que a do projeto Cachalote. Esse fato deve-se ao fato de que no início do projeto Manhanah os desenvolvedores tiveram que desenvolver funcionalidades com lógica complexa sem o apoio dos testes automatizados e consequentemente do CI/CD para poderem refatorar e melhorar tais funções.

Figura 34 – Gráfico Comparativo de Complexidade Ciclomática



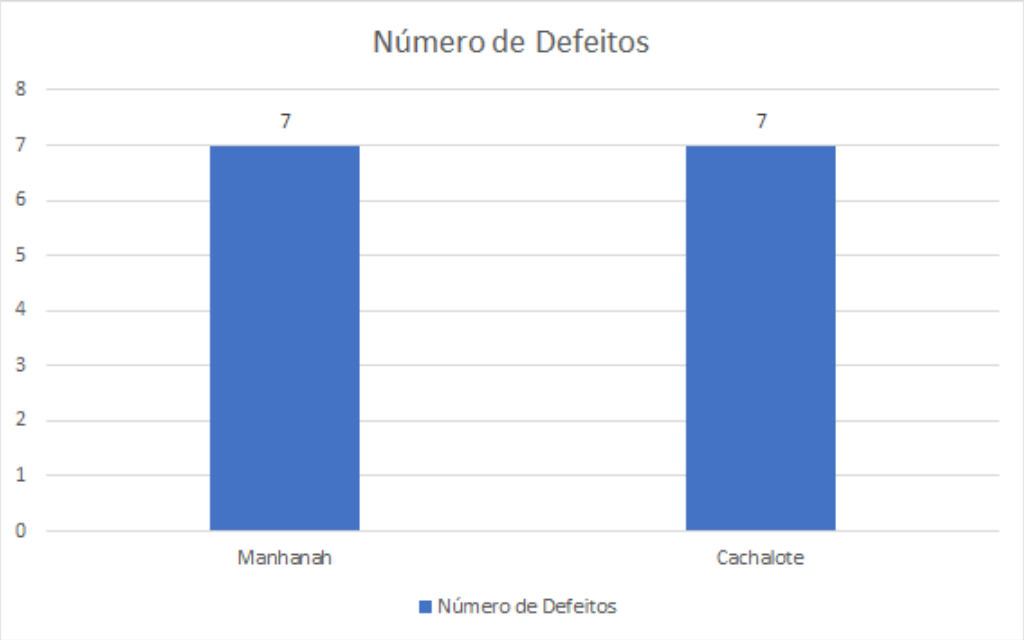
Fonte: Elaborado pelo autor (2022)

Porém ao analisarmos as últimas 9 (nove) execuções do pipeline CI/CD do projeto Manhanah, podemos verificar que a complexidade ciclomática manteve-se estável.

6.4.3 Questão 3: Qual o impacto da adoção de CI/CD na qualidade do produto de software?

M9. Defeitos. Com a utilização da integração e entrega contínua, foram implementados testes automatizados que são sempre executados durante o pipeline. Dessa forma, o software não é disponibilizado até que todos os defeitos encontrados pelos testes automatizados tenham sido corrigidos. A Figura 35 apresenta um gráfico comparativo em relação ao número de defeitos. A quantidade de defeitos dos dois projetos encontra-se igualada, devido ao pouco uso do sistema desenvolvido no projeto Cachalote, enquanto no sistema do projeto Manhanah há uma alta demanda no início e término de cada semestre.

Figura 35 – Gráfico Comparativo do Número de defeitos



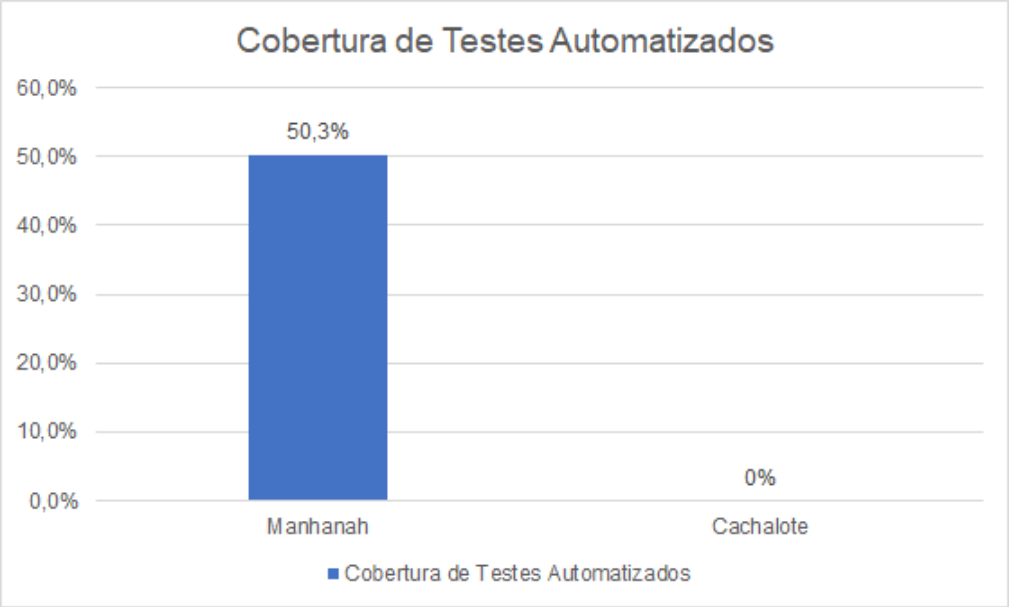
Fonte: Elaborado pelo autor (2022)

M10. Cobertura de Testes Automatizados. O percentual de cobertura dos projetos foi coletado utilizando a biblioteca Covarage.py[23]. Ela monitora o programa, observando quais partes do código foram executadas e, em seguida, analisa a fonte para identificar o código que poderia ter sido executado, mas não foi.

A Figura 36 apresenta um gráfico comparativo com os dados coletados referentes a cobertura de testes automatizados de cada projeto. Como mencionado anteriormente, a COSIS não possui cultura de testes, ou seja, nos projetos desenvolvidos não há testes automatizados, por consequência, a cobertura de teste é 0. Embora no projeto Manhanah, a cobertura de testes automatizados chegue a 50,3%, está muito abaixo do mínimo recomendado por Humble e Farley (2014) e Chen (2018), que estabelecem o mínimo de 75% e 90% respectivamente.

[23] *Coverage.py* é uma ferramenta para medir a cobertura de código de programas Python. Disponível em: <https://coverage.readthedocs.io/>

Figura 36 – Gráfico Comparativo do Percentual de Cobertura de Testes Automatizados



Fonte: Elaborado pelo autor (2022)

6.5 DISCUSSÃO

A seguir encontram-se resumidos os resultados da experiência para adotar integração e entrega contínua.

Este estudo mostrou a utilização da abordagem para adotar CI/CD em um Instituto Federal (IF). O uso e adoção de CI/CD se deu em um projeto piloto do IFAC, após a adoção foi realizada uma avaliação do impacto dessas práticas no processo de entrega e na qualidade do código e do produto de software e uma entrevista com os os desenvolvedores dos dois projetos afim de coletar suas impressões com relação a adoção do pipeline CI/CD.

O esforço investido na adoção e avaliação do CI/CD foi grande, ou seja, 6 (seis) meses. Durante esse período, surgiram alguns desafios, que com o auxilio da abordagem de gestão de risco, todos foram solucionados. Obviamente, a aplicação das práticas de CI/CD em uma avaliação da viabilidade pode ser melhorada e a abordagem para adotá-las pode ser estruturada de forma mais completa para abranger de forma ideal o uso dessas práticas nos órgãos da Administração Pública Federal (APF). Mas, essas melhorias ainda podem ser realizadas em projetos futuros, quando estudos de caso forem realizados em outros órgãos públicos.

Após a conclusão do projeto piloto, o processo de adoção preliminar foi atualizado, de forma a contemplar as experiências obtidas durante a execução desse projeto. Portanto, foi adiantada a tarefa de definição das etapas do pipeline, modificada a ordem da escolha e definição das

melhores práticas e ferramentas e foram acrescentadas 3 (três) etapas: (i) definição do fluxo de trabalho; (ii) definição do processo de entrega; e (iii) avaliação do uso do CI/CD.

A adoção das práticas contínuas trouxe diversas mudanças, não só para o projeto piloto, como também para toda a equipe de desenvolvimento do IFAC. No Quadro 15 está sumarizada as mudanças que a adoção do CI/CD trouxe.

Quadro 15 – Comparativo de antes e depois do CI/CD

Prática	**Antes**	**Depois**
Fluxo de Trabalho de desenvolvimento usando sistema de controle de versão	Inexistente	Utilizado o TBD
Processo de Compilação	Manual	Automatizado utilizando o *Makefile* e Fabric
Testes de Unidade	Inexistente	Automatizado utilizando o Django
Testes de Integração	Inexistente	Automatizado utilizando o Django
Análise Estática	Inexistente	Realizada em cada commit utilizando o SonarQube
Testes de Aceitação	Manual	Automatizado utilizando o *django-behave*
Processo de Implantação	Manual	Automatizado utilizando a biblioteca Fabric
Cycle time	Média de 4 dias	Média de 4h
Tempo de Implantação	Média de 30min	Média de 10min
Métricas	Inexistente	Métricas de qualidade do código coletadas utilizando o SonarQube
Monitoramento de *bugs* e erros	Inexistente	Realizado através de softwares de monitoramento de aplicações e rastreamento de erros

Fonte: Elaborado pelo autor (2022)

Por fim, foram realizadas entrevistas com a equipe dos dois projetos, visando captar as impressões deles com relação as práticas de CI/CD adotas no instituto. Quando questionados se o CI/CD, contribuía positivamente ao projeto, eles elencaram os benefícios da automação e da execução dos testes automatizados, o que faz com que sejam descobertos menos erros em produção e durante a implantação.

Os participantes também relataram que um empecilho para adotar as práticas contínuas é a complexidade da configuração do pipeline. Contudo, 3 dos 4 desenvolvedores se sentiram muito satisfeitos com adoção das práticas contínuas.

6.6 SÍNTESE DO CAPÍTULO

Uma análise preliminar do processo de desenvolvimento e entrega de software, revelou os antipadrões da entrega de versão descritos por Humble e Farley (2014). Portanto, observou-se que a entrega do produto de software era custosa e não repetível, devido ao excesso de erros que ocorria durante o processo de implantação, acarretando em correções frequentes, gerando uma demora para fazer a entrega e a implantação sem garantia da qualidade do produto, devido a um alto número de erros que são reportados tardiamente, gerando atrasos e consequentemente a perca dos prazos além de uma baixa produtividade do setor.

Assim, a integração e a entrega contínua surgiram como uma alternativa viável para a solução dos problemas apresentados nos processos da COSIS, principalmente por propor uma redução do tempo de lançamento do produto, segurança e qualidade nas entregas. Essas práticas foram adotadas em um projeto piloto durante a fase de desenvolvimento, utilizando a abordagem construída no Capítulo 5, e avaliada durante o início da fase de sustentação do projeto, através do GQM, utilizando o método de comparação do projeto irmão que serviu de linha base.

O estudo mostrou que a abordagem auxilia na adoção da integração e entrega contínua. Nele foi relatado a experiência vivenciada durante o processo de adoção, elencando desafios e lições aprendidas, mas, por outro lado, as ferramentas utilizadas dificultaram a obtenção de dados estatisticamente significativos. A partir desse último, verificou-se a necessidade da utilização de uma ferramenta de gerenciamento de projetos e tarefas, mesmo que esta seja paga, visto os benefícios que seriam colhidos.

Por fim, o processo de adoção preliminar foi atualizado, baseando-se nas lições aprendidas e desafios vivenciados, demonstrando que a abordagem e a estratégia auxiliam na adoção de CI/CD principalmente em outros IFs.

No capítulo seguinte, apresentam-se as considerações finais desta pesquisa, com as conclusões a partir dos estudos e análises realizados e as contribuições para o aprofundamento deste campo de estudo.

7 CONCLUSÃO

O Capítulo 7, apresenta as considerações finais deste estudo e está estruturado conforme as seguintes seções:

1. **Síntese dos Resultados** (seção 7.1): esta seção apresenta um resumo dos resultados obtidos em todo trabalho, respondendo a questão de pesquisa, apresentar se hipóteses de pesquisas foram confirmadas e se os objetivos foram alcançados;
2. **Limitações e Ameaças à Validade** (seção 7.2): esta seção apresenta uma discussão sobre as limitações do estudo que a pesquisa sofreu e ameaças à validade;
3. **Trabalhos Futuros** (seção 7.3): apresenta propostas para novas pesquisas;
4. **Considerações Finais** (seção 7.4): apresenta as conclusões obtidas com o trabalho e as lições aprendidas.

7.1 SÍNTESE DOS RESULTADOS

O objetivo geral desta pesquisa foi realizar um estudo sobre a adoção da integração e entrega contínua (CI/CD) quanto ao ciclo de entrega do desenvolvimento de software na DSGTI do IFAC, de forma a verificar o impacto dessa adoção no processo de entrega e qualidade do produto. Para isso, buscou compreender: **"Como integração e a entrega contínua podem ser utilizadas no IFAC afim de reduzir o tempo de inserção no mercado (*time to market*) e o número de erros dos novos recursos do produto de software?"**

Como o intuito de atingir o objetivo traçado, foi realizada uma Revisão Sistemática da Literatura (RSL), na qual buscou-se investigar **"o que muda no desenvolvimento de software quando são implantadas práticas de integração/entrega contínua?"** e **"como apoiar o uso dessas práticas?"**. Para orientar a análise dos dados, foram elaboradas três questões mais específicas, descritas a seguir:

- **Q1:** Quais desafios foram relatados para a adoção de práticas contínuas?
- **Q2:** Que práticas foram relatadas para implementar com êxito práticas contínuas?

- **Q3:** Quais ferramentas foram empregadas para projetar e implementar pipelines de implantação?

A RSL selecionou 54 estudos relevantes ao contexto, em 4 bases de dados consultadas, os quais juntamente com outras informações integraram a fundamentação da proposta do processo de adoção de pipeline CI/CD e da abordagem para o gerenciamento de risco do pipeline.

O questionamento **Q1** buscou investigar os principais desafios em relação a adoção de um pipeline de implantação. Como resultado, foram identificados pela pesquisa 32 desafios, que podem ser vistos no Apêndice C. Os desafios foram agrupados em 9 categorias, sendo elas: Humano e Organizacional, Processo, Ferramentas, Design de Compilação,Integração, Arquitetura da Aplicação, Teste e Qualidade, Entrega, Infraestrutura e Monitoramento.

O questionamento **Q2** buscou encontrar boas práticas adotadas para implementar práticas contínuas. A partir das evidências coletadas pela revisão sistemática, 20 Melhores práticas a serem adotadas no pipeline de implantação foram identificadas, que podem ser vistas no Apêndice D.

O questionamento **Q3** buscou encontrar as ferramentas de implantação e as ferramentas que são usadas no pipeline de implantação. Como resultado, foram encontradas 63 ferramentas de apoio para adoção de pipelines de implantação, que podem ser vistas no Apêndice E. As ferramentas foram categorizadas em 9 categorias: (i) gerenciamento de projetos; (ii) Sistema de Controle de Versão, em inglês *Version Control Systems* (VCS); (iii) ferramenta de gerenciamento e análise de código; (iv) construção do software (*build*); (v) servidor de integração contínua; (vi) ferramenta de teste; (vii) repositório de artefatos; (viii) configuração e provisionamento; e (ix) monitoramento.

Com o resultado da revisão sistemática, foi proposto um processo para a adoção de um pipeline CI/CD e uma abordagem para o gerenciamento de risco do pipeline. O processo foi construído tendo como base o processo descrito por Duvall, Matyas e Glover (2007) e Humble e Farley (2014).

A abordagem foi construída tendo como base o Método de Análise e Solução de Problemas (MASP), usando evidências da experimentação científica e industrial, para refinar uma teoria de gerenciamento do pipeline CI/CD que pode ser construída gradualmente e de forma colaborativa. Para ajudar na identificação dos desafios por parte da equipe de desenvolvimento, as evidências oriunda da RSL relacionadas a desafios, melhores práticas e ferramentas foram

combinadas (ver Tabela 3, Tabela 4, Tabela 5) de forma que, diante da identificação de um desafio, possam verificar as possibilidades que venha a minimizar ou eliminar os obstáculos durante a adoção do CI/CD.

O processo criado nesta pesquisa foi aplicado em um projeto piloto do IFAC. Antes de adotar o CI/CD, foi realizado um estudo preliminar do processo de entrega de software, relatando o histórico sobre o desenvolvimento de software no instituto, os processos atuais de desenvolvimento e entrega de software e, por fim, uma avaliação dos processos.

O processo de adoção foi documentado nesta pesquisa através de um relato de experiência, apresentando estratégias minímas para a adoção do pipeline CI/CD em uma instituição de ensino pública, contendo as etapas percorridas, ferramentas utilizadas, desafios superados e lições aprendidas. A avaliação do processo de adoção se deu através do GQM utilizando o método de comparação do projeto irmão, que envolve dois projetos, um que utiliza o novo método e outro que utiliza o método atual, no caso dessa pesquisa, o novo método era o pipeline CI/CD e o método atual era a implantação manual.

Além da avaliação do processo através da coleta de métricas, foram realizadas entrevistas com a equipe dos dois projetos, visando captar as impressões deles com relação as práticas de CI/CD adotas no instituto. Os participantes relataram que um empecilho para adotar as práticas contínuas é a complexidade da configuração do pipeline e destacaram os benefícios da automação e da execução dos testes. Mesmo com o empecilho elencado pela equipe, 3 dos 4 desenvolvedores se sentiram muito satisfeitos com adoção das práticas contínuas.

Conforme pode ser visto a seguir, esta pesquisa comprovou as hipóteses de pesquisa através do relato de experiência e da avaliação da adoção do CI/CD quanto à viabilidade apresentada na seção 6.4.

- **H1. Adotar integração e entrega contínua no processo de desenvolvimento e entrega do produtos de software dos IF reduz o tempo em retrabalhos e aumenta a qualidade do produto:** com a adoção do CI/CD, verificou-se que o número de *bugs*, vulnerabilidades e defeitos foram reduzindo e a cobertura de testes aumentando, diminuindo assim o retrabalho, reduzindo a execução de tarefas manuais, liberando o desenvolvedor realizar outras tarefas;

- **H2. O comprometimento dos envolvidos com a adoção de um pipeline de integração e entrega contínua traz benefícios para a equipe e para o projeto:** como pode ser visto com a adoção do CI/CD há uma maior previsibilidade no tempo

de implantação, que embora seja 8min, nesse tempo o responsável pelo processo de implantação pode realizar o monitoramento do processo e usar seu tempo em outra atividade.

Diante do exposto, conclui-se que, assim como o objetivo principal desta pesquisa foi alcançado, com as estratégias minímas para a adoção do pipeline através do relato de experiência, todos os objetivos secundários propostos no início da pesquisa (seção 1.4) foram igualmente cumpridos de forma satisfatória.

7.2 LIMITAÇÕES E AMEAÇAS À VALIDADE

Um dos pontos fortes da abordagem de adoção e gestão de risco de um pipeline CI/CD proposta neste trabalho é que ela é baseada em evidências existentes no meio industrial e científico. Uma vez que estas evidências foram encontradas em artigos de revistas de qualidade, periódicos, conferências, etc, espera-se que os mesmos tenham sido avaliados quanto às ameaças a validade. Porém, a abordagem herda as ameaças à validade dos estudos a partir do qual as evidências foram coletadas. Para superar essa fragilidade inevitável, foi aplicada essa abordagem em um projeto piloto, no entanto, se faz necessário a realização de estudos de caso ou pesquisa-ação para avaliar a abordagem. Além disso, apesar da preocupação em utilizar um quadro metodológico rigoroso, esta pesquisa possui algumas limitações:

- Uma análise da distribuição geográfica dos estudos experimentais e dos relatórios de experiência industrial não foi realizada, porque a maioria dos estudos não fornece informações contextuais sobre o lugar onde foram realizados. Portanto, não é possível saber se os estudos se concentram em determinadas regiões ou países. Isso representa uma ameaça à validade externa, uma vez que não é possível saber se os resultados representam todos os principais desenvolvedores de software ao redor do mundo;

- Devido a restrições de tempo e orçamento, a pesquisa não considerou algumas bases de dados que são usadas por Shahin, Ali Babar e Zhu (2017), Proulx et al. (2018) e Laukkanen, Itkonen e Lassenius (2017): *SpringerLink*, *Wiley Online Library* e *ISI Web of Science*. Embora isso possa representar uma limitação e uma ameaça a validade, as principais conferências da área e revistas foram pesquisadas, reduzindo o problema. O

protocolo de revisão pode ser usado para estender os resultados usando essas e outras fontes;

- Além disso, apenas o processo de busca foi revisado por mais de um pesquisador, como é sugerido pelos guias. A maior parte da extração foi realizada por apenas um pesquisador. Isso representa uma limitação, mas é previsto por Kitchenham (2007) para alunos de PhD que utilizam revisão sistemática (nada foi dito sobre alunos de mestrado). Segundo a autora, basta que o orientador da tese, assim como outros envolvidos, participe da revisão do protocolo e revise partes da execução da revisão. Neste sentido, o orientador revisou todo o protocolo de estudo deste trabalho, bem como a síntese dos dados e o relatório de resultados;

- Embora tenha sido utilizada a abordagem em um projeto piloto do IFAC, este estudo da adoção, apresentou estratégia mínima viável para adoção de integração e entrega contínua através de um relato de experiência e lições aprendidas e uma avaliação do uso dessas práticas. No entanto, estudos primários devem ser realizados para testar a abordagem.

- Embora a avaliação de qualidade dos estudos tenha sido realizada por apenas um pesquisador com pouca experiência, o protocolo foi revisado pelo orientador. A elaboração do formulário de avaliação focou na identificação da metodologia, objetivos, justificativa, contexto da pesquisa e se o estudo estava bem referenciado, bem como se o estudo respondia as questões de pesquisa da revisão. Com o formulário simplificado a execução da etapa de avaliação da qualidade do estudo ficou facilitada, demonstrando a relevância dos estudos no contexto desta pesquisa.

- Para evitar preconceitos e garantir a validade interna na avaliação de um novo método, se faz necessário identificar uma base válida para avaliar os resultados do estudo (KITCHENHAM; PICKARD; PFLEEGER, 1995). Portanto, foi escolhido o método de comparação do projeto irmão que serviu como linha de base. Este método, envolve dois projetos, o projeto piloto que utiliza o novo método e o projeto irmão que utiliza o método atual. Os projetos escolhidos tiveram como produto o desenvolvimento de aplicações web utilizando Python com o *framework* Django, com tamanhos semelhantes, além de suas importâncias no contexto acadêmico;

- De acordo com Kitchenham, Pickard e Pfleeger (1995), quando o efeito de um fator não pode ser devidamente distinguido do efeito de outro fator, os dois fatores são confundidos. No caso do estudo sobre a adoção e uso do pipeline CI/CD, os efeitos de aprender a usar uma nova abordagem, neste caso práticas de integração e entrega contínua, podem interferir nos benefícios de usá-la. Por exemplo, o esforço da utilização de uma abordagem A devido a curva de aprendizado, pode acarretar em um aumento do tempo de desenvolvimento de software e, consequentemente, levar a uma diminuição da produtividade. Para evitar o efeito da redução da produtividade, foram separadas as atividades destinadas a aprender a usar as práticas de integração e entrega contínua, daquelas destinadas a avaliá-la.

7.3 TRABALHOS FUTUROS

As limitações acima referidas oferecem caminhos claros para novas pesquisas.

- Realização de um estudo afim de investigar as diferenças práticas entre o DevOps e GitOps. Segundo Beetz e Harrer (2022), o GitOps foi introduzido pela empresa Weaveworks, como um modelo para operar *clusters* Kubernetes e aplicativos nativos da nuvem em geral, usando o sistema de controle de versão Git como fonte única de verdade. O GitOps está sendo "cotado" como a evolução do DevOps. Portanto, o estudo poderia realizar uma combinação das duas metodologias ou substituir DevOps por GitOps;

- Realização de um estudo para investigar a utilização de SRE juntamente com DevOps;

- Utilização prática da abordagem, através da condução de estudos de caso ou pesquisa-ação em ambientes industriais, é necessária não só para testar a abordagem, mas também aumentar o número de evidências sobre a utilização de boas práticas, ferramentas na adoção de um pipeline de integração e entrega contínua;

- A abordagem também pode ser refinada, aumentando-se o número de estudos primários analisados na revisão sistemática. Para isso, outros pesquisadores, utilizando o protocolo definido para este trabalho podem replicar o estudo e comparar os resultados;

- Por fim, podem ser realizados estudos específicos para alguns dos desafios que foram muito citados pelas evidências. Com um trabalho mais direcionado para um ou mais desafios, novas boas práticas e ferramentas podem ser identificados.

7.4 CONSIDERAÇÕES FINAIS

Esta pesquisa buscou além de contribuir com o desenvolvimento e entrega de software, contribuir com as pesquisas em Engenharia de Software com foco em práticas contínuas. O trabalho apresentou um processo detalhado para a realização de uma revisão sistemática da literatura que pode ser seguido por outros pesquisadores para outros trabalhos. Além de um corpo de conhecimento, a partir das evidências coletadas como resultado dessa revisão, que funciona como um *framework* para compor o pipeline concreto e efetivo a ser implementado, um processo de adoção de pipeline CI/CD e uma abordagem para gerenciamento de risco do pipeline utilizando o corpo de conhecimento. Por fim, este trabalho também contribui apresentando estratégias mínimas para adotar o pipeline de integração e entrega contínua, através do relato de experiência da adoção e uso dessas práticas em um projeto piloto do IFAC, o qual apenas deu inicio a adoção dessas práticas, que atualmente conta com metade dos projetos desenvolvidos pelo IFAC fazendo o uso da integração contínua.

Algumas lições aprendidas neste trabalho são listadas abaixo, como forma de ajudar outros pesquisadores em trabalhos futuros:

- Uma revisão sistemática extensiva exige tempo e dedicação. Para este trabalho, o protocolo começou a ser desenvolvido em julho de 2019. E o processo de revisão teve duração total de 14 meses. Isso deve ser levando em consideração no planejamento;

- O protocolo de pesquisa deve ser claro e objetivo, não deixar margem para mal entendidos. Com a definição dos principais pontos, o trabalho pode ser iniciado, e o protocolo paralelamente ir evoluindo;

- A sintaxe das *strings* precisa ser adaptada para a realidade das pesquisas em diferentes fontes, para identificar as particularidades de cada fonte, pode-se realizar testes. Além disso, muitas fontes de pesquisa precisam de uma licença especial para visualização e *download* dos trabalhos;

- As buscas retornam muitos trabalhos irrelevantes para o escopo da pesquisa, isso deve ser levando em consideração e os critérios de inclusão/exclusão devem garantir a imparcialidade na seleção dos estudos primários;

- A definição dos critérios de qualidade é um dos maiores desafios na definição do protocolo e em muitos trabalhos é deixado de lado. Deve haver uma preocupação com essa questão, já que a avaliação pode dar uma maior relevância aos resultados da pesquisa;

- O uso de ferramentas computacionais para a extração e síntese dos dados melhora o tempo de trabalho e determinadas ferramentas podem proporcionar uma melhor visão dos dados extraídos e relações entre eles;

- Propor a integração e entrega contínua como um analgésico. Usando essa estratégia, identifica-se os pontos problemáticos de cada parte interessada e depois seleciona-se aqueles que os CI/CD podem ajudar a resolver. Ao serem apresentados os CI/CD para a parte interessada, deve-se concentrar em explicar como e porque o CI/CD pode ajudar a resolver os pontos problemáticos identificados;

- Designar um responsável pela implantação de CI/CD de fora do projeto, pois caso seja um membro da equipe de desenvolvimento sua atenção fica dividida entre a configuração e desenvolvimento do projeto;

- Adotar um sistema para o gerenciamento de projetos/tarefas, uma vez que usar o *issue tracker* nativo das ferramentas Git pode prejudicar a visualização do andamento das tarefas pela equipe do projeto;

- Avaliar o novo método fornece bases para saber qual o impacto de sua adoção. A abordagem de medição GQM auxiliou a planejar e estabelecer um sistema de medição que estivesse baseado em objetivos específicos, proporcionando um foco durante o processo de avaliação;

- Estabelecer métodos de comparação para o novo método a ser implantado, afim de evitar vieses. Kitchenham, Pickard e Pfleeger (1995) estabelecem três métodos de comparação: (i) projeto irmão; (ii) linha de base da empresa; e (iii) se o método se aplica a componentes individuais, aplique-o aleatoriamente a alguns componentes do produto e não a outros.

Finalmente, os resultados desta pesquisa contribuem para a adoção e utilização de integração e entrega contínua de duas maneiras. Primeiro, os resultados da revisão sistemática fornecem à comunidade acadêmica uma melhor compreensão sobre os desafios na adoção de um pipeline CI/CD e, com isso, mostram lacunas na área que podem ser boas oportunidades para futuras pesquisas. Segundo, a abordagem para o adoção e gerenciamento de risco do pipeline CI/CD aliada ao relato de experiência da adoção e uso dessas práticas pode apoiar os profissionais e pesquisadores na identificação de desafios relevantes e definição de soluções para os mesmos, utilizando para isso, as melhores práticas, ferramentas que já foram testados por outros estudos primários, em ambientes experimentais e industriais, bem como auxiliar a quem quiser adotar tais práticas.

REFERÊNCIAS

BARBOZA, H. *Limites da aplicação da metodologia ágil no setor público*. Tese (Doutorado) — Escola Brasileira de Administração Pública e de Empresas, Centro de Formação Acadêmica e Pesquisa, Rio de Janeiro, 2019.

BASILI, V. R.; CALDIERA, G.; ROMBACH, H. D. The goal question metric approach. *Encyclopedia of software engineering*, p. 528–532, 1994.

BASS, L.; WEBER, I.; ZHU, L. *DevOps: A software architect's perspective*. [S.l.]: Addison-Wesley Professional, 2015.

BEECHAM, S.; BADDOO, N.; HALL, T.; ROBINSON, H.; SHARP, H. Motivation in software engineering: A systematic literature review. *Information and software technology*, Elsevier, v. 50, n. 9-10, p. 860–878, 2008.

BEETZ, F.; HARRER, S. Gitops: The evolution of devops? *IEEE Software*, v. 39, n. 4, p. 70–75, 2022.

BERNARDO, J. a. H.; COSTA, D. A. da; KULESZA, U. Studying the impact of adopting continuous integration on the delivery time of pull requests. In: *Proceedings of the 15th International Conference on Mining Software Repositories*. New York, NY, USA: Association for Computing Machinery, 2018. (MSR '18), p. 131–141. ISBN 9781450357166. Disponível em: <https://doi.org/10.1145/3196398.3196421>.

BEYER, B.; JONES, C.; PETOFF, J.; MURPHY, N. R. *Site Reliability Engineering: How Google Runs Production Systems*. Sebastopol, CA: O'Reilly Media, Inc., 2016. Disponível em: <https://sre.google/sre-book/table-of-contents/>.

BEYER, B.; MURPHY, N. R.; RENSIN, D. K.; KAWAHARA, K.; THORNE, S. *The Site Reliability Workbook: Practical Ways to Implement SRE*. Sebastopol, CA: O'Reilly Media, Inc., 2018. Disponível em: <https://sre.google/workbook/table-of-contents/>.

CHEN, L. Continuous delivery: Overcoming adoption challenges. *Journal of Systems and Software*, v. 128, p. 72–86, 2017. ISSN 0164-1212. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0164121217300353>.

CHEN, L. Microservices: Architecting for continuous delivery and devops. In: *2018 IEEE International Conference on Software Architecture (ICSA)*. [S.l.: s.n.], 2018. p. 39–397.

COIS, C. A.; YANKEL, J.; CONNELL, A. Modern devops: Optimizing software development through effective system interactions. In: *2014 IEEE International Professional Communication Conference (IPCC)*. [S.l.: s.n.], 2014. p. 1–7.

CRUZ, V. L.; ALBUQUERQUE, A. B. A devops introduction process for legacy systems. In: *2018 XLIV Latin American Computer Conference (CLEI)*. [S.l.: s.n.], 2018. p. 139–148. ISSN null.

DRIESSEN, V. *A successful Git branching model*. 2010. Disponível em: <https://nvie.com/posts/a-successful-git-branching-model/>. Online; Acesso em: 24 set. 2020.

DUVALL, P. M.; MATYAS, S.; GLOVER, A. *Continuous integration: improving software quality and reducing risk*. [S.l.]: Pearson Education, 2007.

EASTERBROOK, S.; ARANDA, J.; PERRY, D. E.; SIM, S. E. Case studies for software engineers. In: *Proceedings of the 28th International Conference on Software Engineering*. New York, NY, USA: Association for Computing Machinery, 2006. (ICSE '06), p. 1045–1046. ISBN 1595933751. Disponível em: <https://doi.org/10.1145/1134285.1134497>.

EASTERBROOK, S.; SINGER, J.; STOREY, M.-A.; DAMIAN, D. Selecting empirical methods for software engineering research. In: SHULL, F.; SINGER, J.; SJøBERG, D. I. K. (Ed.). *Guide to advanced empirical software engineering*. Springer, 2008. p. 285–311. Disponível em: <https://doi.org/10.1007/978-1-84800-044-5_11>.

ELBERZHAGER, F.; ARIF, T.; NAAB, M.; Süß, I.; KOBAN, S. From agile development to devops: Going towards faster releases at high quality – experiences from an industrial context. In: WINKLER, D.; BIFFL, S.; BERGSMANN, J. (Ed.). *Software Quality. Complexity and Challenges of Software Engineering in Emerging Technologies*. Cham: Springer International Publishing, 2017. p. 33–44. ISBN 978-3-319-49421-0. Disponível em: <https://doi.org/10.1007/978-3-319-49421-0_3>.

FITZGERALD, B.; STOL, K.-J. Continuous software engineering: A roadmap and agenda. *Journal of Systems and Software*, Elsevier, v. 123, p. 176–189, 2017.

FOWLER, M. *Continuos Integration*. 2006. Disponível em: <https://www.martinfowler.com/articles/continuousIntegration.html>. Online; Acesso em: 15 dez. 2019.

FOWLER, M. *ContinuousDelivery*. 2013. Disponível em: <https://martinfowler.com/bliki/ContinuousDelivery.html>. Online; Acesso em: 7 abr. 2022.

FOWLER, M. *Refactoring: improving the design of existing code*. [S.l.]: Addison-Wesley Professional, 2018.

GERMANI, L. et al. Desafios para o desenvolvimento de serviços digitais pelo governo federal brasileiro. *Cultura Digital, internet e apropriações políticas*, p. 73, 2016.

GOMEDE, E.; SILVA, R. T. D.; BARROS, R. M. Relato de experiência sobre a implantação de um processo de entrega contínua em uma organização da indústria financeira. In: *CIbSE*. [S.l.: s.n.], 2015. p. 724.

HAMMANT, P.; SMITH, S. *Trunk-Based Development and Branch By Abstraction*. 2017. Disponível em: <https://trunkbaseddevelopment.com/>. Online; Acesso em: 24 set. 2020.

HUMBLE, J. Continuous delivery sounds great, but will it work here? *Commun. ACM*, Association for Computing Machinery, New York, NY, USA, v. 61, n. 4, p. 34–39, mar 2018. ISSN 0001-0782. Disponível em: <https://doi.org/10.1145/3173553>.

HUMBLE, J.; FARLEY, D. *Entrega contínua: como entregar software de forma rápida e confiável*. [S.l.]: Porto Alegre: Bookman, 2014.

HUMBLE, J.; MOLESKY, J. Why enterprises must adopt devops to enable continuous delivery. *Cutter IT Journal*, v. 24, n. 8, p. 6–12, aug 2011.

IBRAHIM, M. M. A.; SYED-MOHAMAD, S. M.; HUSIN, M. H. Managing quality assurance challenges of devops through analytics. In: *Proceedings of the 2019 8th International Conference on Software and Computer Applications*. New York, NY, USA: ACM, 2019. (ICSCA '19), p. 194–198. ISBN 978-1-4503-6573-4. Disponível em: <http://doi.acm.org/10.1145/3316615.3316670>.

IFAC, I. F. d. A. *Histórico*. 2016. Disponível em: <https://portal.ifac.edu.br>. Online; Acesso em: 15 dez. 2019.

IVANOV, V.; SMOLANDER, K. Implementation of a devops pipeline for serverless applications. In: KUHRMANN, M.; SCHNEIDER, K.; PFAHL, D.; AMASAKI, S.; CIOLKOWSKI, M.; HEBIG, R.; TELL, P.; KLÜNDER, J.; KÜPPER, S. (Ed.). *Product-Focused Software Process Improvement*. Cham: Springer International Publishing, 2018. p. 48–64. ISBN 978-3-030-03673-7.

JOHNSON, S.; MURPHY, A. *The Works of Samuel Johnson, LL.D.* [S.l.]: Jones & Companey, 1825. Vol. II. (The Works of Samuel Johnson, LL.D, Vol. II).

KIM, G.; HUMBLE, J.; DEBOIS, P.; WILLIS, J. *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations*. Portland: IT Revolution Press, 2016. (ITpro collection). ISBN 9781942788072.

KITCHENHAM, B. *Guidelines for performing Systematic Literature Reviews in Software Engineering*. Durham, UK, 2007. V. 2.3.

KITCHENHAM, B. A.; PICKARD, L. M. Evaluating software engineering methods and tools: Part 9: Quantitative case study methodology. *SIGSOFT Softw. Eng. Notes*, Association for Computing Machinery, New York, NY, USA, v. 23, n. 1, p. 24–26, jan. 1998. ISSN 0163-5948. Disponível em: <https://doi.org/10.1145/272263.272268>.

KITCHENHAM, B. A.; PICKARD, L. M.; PFLEEGER, S. L. Case studies for method and tool evaluation. *IEEE Software*, v. 12, n. 4, p. 52–62, 1995.

LAUKKANEN, E.; ITKONEN, J.; LASSENIUS, C. Problems, causes and solutions when adopting continuous delivery — a systematic literature review. *Information and Software Technology*, Elsevier, v. 82, p. 55–79, feb 2017. ISSN 0950-5849. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0950584916302324>.

LOPES, M. D. *Aplicação da integração contínua em um sistema de auxílio à tomada de decisão*. Dissertação (Mestrado) — Programa de Pós-Graduação em Computação Aplicada, Universidade de Passo Fundo, Passo Fundo, RS, 2020. Disponível em: <http://tede.upf.br/jspui/handle/tede/1902>.

LWAKATARE, L. E.; KARVONEN, T.; SAUVOLA, T.; KUVAJA, P.; OLSSON, H. H.; BOSCH, J.; OIVO, M. Towards devops in the embedded systems domain: Why is it so hard? In: *2016 49th Hawaii International Conference on System Sciences (HICSS)*. [S.l.: s.n.], 2016. p. 5437–5446. ISSN 1530-1605.

MARCONI, M. d. A.; LAKATOS, E. M. *Metodologia científica*. 5. ed. São Paulo: Atlas, 2010.

PESSÔA, S. d. S. Entrega contínua de software: um estudo de caso. *Trabalho de Conclusão de Curso (graduação) — Universidade de Brasília, Faculdade UnB Gama*, Brasília, DF, 2018.

PNP, P. N. P. *PNP 2019*. 2019. Disponível em: <https://www.plataformanilopecanha.org/>. Online; Acesso em: 15 dez. 2019.

PROULX, A.; RAYMOND, F.; ROY, B.; PETRILLO, F. Problems and Solutions of Continuous Deployment: A Systematic Review. dec 2018. Disponível em: <https://arxiv.org/abs/1812.08939http://arxiv.org/abs/1812.08939>.

RILEY, C. *How to Keep CALMS and Release More!* 2014. Disponível em: <https://www.rapid7.com/blog/post/2014/10/24/how-to-keep-calms-and-release-more/>. Online; Acesso em 10 jun. 2021.

RIUNGU-KALLIOSAARI, L.; MÄKINEN, S.; LWAKATARE, L. E.; TIIHONEN, J.; MÄNNISTÖ, T. Devops adoption benefits and challenges in practice: a case study. In: SPRINGER. *International Conference on Product-Focused Software Process Improvement*. [S.l.], 2016. p. 590–597.

RODRÍGUEZ, P.; HAGHIGHATKHAH, A.; LWAKATARE, L. E.; TEPPOLA, S.; SUOMALAINEN, T.; ESKELI, J.; KARVONEN, T.; KUVAJA, P.; VERNER, J. M.; OIVO, M. Continuous deployment of software intensive products and services: A systematic mapping study. *Journal of Systems and Software*, v. 123, p. 263–291, 2017. ISSN 0164-1212. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0164121215002812>.

ROSSI, C.; SHIBLEY, E.; SU, S.; BECK, K.; SAVOR, T.; STUMM, M. Continuous deployment of mobile software at facebook (showcase). In: *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. New York, NY, USA: Association for Computing Machinery, 2016. (FSE 2016), p. 12–23. ISBN 9781450342186. Disponível em: <https://doi.org/10.1145/2950290.2994157>.

RUNESON, P.; HÖST, M. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, v. 14, n. 2, p. 131, Dec 2008. ISSN 1573-7616. Disponível em: <https://doi.org/10.1007/s10664-008-9102-8>.

SHAHIN, M.; ALI BABAR, M.; ZHU, L. Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices. *IEEE Access*, v. 5, p. 3909–3943, 2017. ISSN 2169-3536. Disponível em: <http://ieeexplore.ieee.org/document/7884954/>.

SILVA, B. S.; CARVALHO, G. d.; SANTOS, O. d. Adaptação na prática de um setor público às metodologias ágeis. *Monografia. Ciência da Computação. PUC-RJ*, 2012.

SIQUEIRA, R.; CAMARINHA, D.; WEN, M.; MEIRELLES, P.; KON, F. Continuous delivery: Building trust in a large-scale, complex government organization. *IEEE Software*, v. 35, n. 2, p. 38–43, 2018. Disponível em: <https://doi.org/10.1109/MS.2018.111095426>.

SOUSA, L. F. R. *DevOps: estudo de caso*. Dissertação (Mestrado) — Instituto Politécnico de Coimbra, Instituto Superior de Contabilidade e Administração de Coimbra, Coimbra, 2019. Disponível em: <http://hdl.handle.net/10400.26/31932>.

TRAVASSOS, G.; BIOLCHINI, J. Revisões sistemáticas aplicadas a engenharia de software. In: *XXI SBES-Brazilian Symposium on Software Engineering*. [S.l.: s.n.], 2007.

VERONA, J. *Practical DevOps*. [S.l.]: Packt Publishing Ltd, 2016.

VIRMANI, M. Understanding devops amp; bridging the gap from continuous integration to continuous delivery. In: *Fifth International Conference on the Innovative Computing Technology (INTECH 2015)*. [S.l.: s.n.], 2015. p. 78–82.

VIRTANEN, A.; KUUSINEN, K.; LEPPäNEN, M.; LUOTO, A.; KILAMO, T.; MIKKONEN, T. On continuous deployment maturity in customer projects. In: *Proceedings of the Symposium on Applied Computing*. New York, NY, USA: Association for Computing

Machinery, 2017. (SAC '17), p. 1205–1212. ISBN 9781450344869. Disponível em: <https://doi.org/10.1145/3019612.3019777>.

VISSER, J.; RIGAL, S.; WIJNHOLDS, G.; LUBSEN, Z. *Building Software Teams: Ten Best Practices for Effective Software Development*. [S.l.]: O'Reilly Media, 2017. ISBN 978-1-491-95177-4.

WILLIS, J. *What Devops Means to Me*. 2010. Disponível em: <https://blog.chef.io/what-devops-means-to-me>. Online; Acesso em 07 jun. 2021.

WOHLIN, C.; RUNESON, P.; HÖST, M.; OHLSSON, M. C.; REGNELL, B.; WESSLÉN, A. *Experimentation in Software Engineering*. Springer-Verlag Berlin Heidelberg, 2012. ISBN 978-3-642-29043-5. Disponível em: <https://doi.org/10.1007/978-3-642-29044-2>.

APÊNDICE A – ESTUDOS PRIMÁRIOS

ID	Ano	Fonte	Referência	Qualidade
EP_01	2017	IEEE	Shahin, M; Ali Babar, M.; Zhu, L. Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices. in IEEE Access, vol. 5, pp. 3909-3943, 2017, doi: 10.1109/ACCESS.2017.2685629	Excelente (>86%)
EP_02	2018	IEEE	Arachchi, S. A. I. B. S.; Perera, I. Continuous Integration and Continuous Delivery Pipeline Automation for Agile Software Project Management. 2018 Moratuwa Engineering Research Conference (MERCon), Moratuwa, 2018, pp. 156-161, doi: 10.1109/MERCon.2018.8421965.	Baixa (<26%)
EP_03	2017	IEEE	Mårtensson, T.; Hammarström, P.; Bosch, J. Continuous Integration is Not About Build Systems. 2017 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Vienna, 2017, pp. 1-9, doi: 10.1109/SEAA.2017.30.	Boa (46%-65%)
EP_04	2017	IEEE	Zampetti, F. et al. How Open Source Projects Use Static Code Analysis Tools in Continuous Integration Pipelines. 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR), Buenos Aires, 2017, pp. 334-344, doi: 10.1109/MSR.2017.2.	Muito Boa (66%-85%)
EP_05	2018	IEEE	Cruz, V. L.; Albuquerque, A. B. A. DevOps Introduction Process for Legacy Systems. 2018 XLIV Latin American Computer Conference (CLEI), São Paulo, Brazil, 2018, pp. 139-148, doi: 10.1109/CLEI.2018.00025.	Excelente (>86%)
EP_06	2016	IEEE	Vassallo, C. et al. Continuous Delivery Practices in a Large Financial Organization. 2016 IEEE International Conference on Software Maintenance and Evolution (ICSME), Raleigh, NC, 2016, pp. 519-528, doi: 10.1109/ICSME.2016.72.	Muito Boa (66%-85%)

EP_07	2019	IEEE	Mårtensson, T.; Ståhl, D.; Martini, A.; Bosch, J. Continuous Architecture: Towards the Goldilocks Zone and Away from Vicious Circles. 2019 IEEE International Conference on Software Architecture (ICSA), Hamburg, Germany, 2019, pp. 131-140, doi: 10.1109/ICSA.2019.00022.	Muito Boa (66%-85%)
Ep_08	2017	IEEE	Amrit, C.; Meijberg, Y. Effectiveness of Test-Driven Development and Continuous Integration: A Case Study. in IT Professional, vol. 20, no. 1, pp. 27-35, January/February 2018, doi: 10.1109/MITP.2018.014121554.	Boa (46%-65%)
EP_09	2018	IEEE	Häkli, A.; Taibi, D.; Systa, K. Towards Cloud Native Continuous Delivery: An Industrial Experience Report. 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion), Zurich, 2018, pp. 314-320, doi: 10.1109/UCC-Companion.2018.00074.	Muito Boa (66%-85%)
EP_10	2019	IEEE	Vasile, T.; Cane, S.; Bertram, C.; Jakob, F. Applying Security Concepts to Continuous Integration for the Purpose of Testing Embedded Systems. AmE 2019 - Automotive meets Electronics; 10th GMM-Symposium, Dortmund, Germany, 2019, pp. 1-6.	Boa (46%-65%)
EP_11	2019	IEEE	Singh, C.; Gaba, N. S.; Kaur, M.; Kaur, B. Comparison of Different CI/CD Tools Integrated with Cloud Platform. 2019 9th International Conference on Cloud Computing, Data Science & Engineering (Confluence), Noida, India, 2019, pp. 7-12, doi: 10.1109/CONFLUENCE.2019.8776985.	Boa (46%-65%)
EP_12	2019	IEEE	Heistand, C. et al. DevOps for Spacecraft Flight Software. 2019 IEEE Aerospace Conference, Big Sky, MT, USA, 2019, pp. 1-16, doi: 10.1109/AERO.2019.8742143.	Muito Boa (66%-85%)
EP_13	2017	IEEE	Parnin, C. et al. The Top 10 Adages in Continuous Deployment. in IEEE Software, vol. 34, no. 3, pp. 86-95, May-Jun. 2017, doi: 10.1109/MS.2017.86.	Muito Boa (66%-85%)

EP_14	2017	IEEE	Shahin, M.; Ali Babar M.; Zahedi, M.; Zhu, L. Beyond Continuous Delivery: An Empirical Investigation of Continuous Deployment Challenges. 2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), Toronto, ON, 2017, pp. 111-120, doi: 10.1109/ESEM.2017.18.	Excelente (>86%)
EP_15	2016	IEEE	Callanan, M.; Spillane, A. DevOps: Making It Easy to Do the Right Thing. in IEEE Software, vol. 33, no. 3, pp. 53-59, May-June 2016, doi: 10.1109/MS.2016.66.	Muito Boa (66%-85%)
EP_16	2016	IEEE	Lwakatare, L. E. et al., Towards DevOps in the Embedded Systems Domain: Why is It So Hard?. 2016 49th Hawaii International Conference on System Sciences (HICSS), Koloa, HI, 2016, pp. 5437-5446, doi: 10.1109/HICSS.2016.671.	Muito Boa (66%-85%)
EP_17	2018	IEEE	Chen, L. Microservices: Architecting for Continuous Delivery and DevOps. 2018 IEEE International Conference on Software Architecture (ICSA), Seattle, WA, 2018, pp. 39-397, doi: 10.1109/ICSA.2018.00013.	Muito Boa (66%-85%)
EP_18	2018	ACM	Sampedro, Zebula; Holt, Aaron; Hauser, Thomas. 2018. Continuous Integration and Delivery for HPC: Using Singularity and Jenkins. In Proceedings of the Practice and Experience on Advanced Research Computing (PEARC '18). Association for Computing Machinery, New York, NY, USA, Article 6, 1–6. DOI:https://doi.org/10.1145/3219104.3219147	Boa (46%-65%)
EP_19	2017	ACM	Virtanen, Antti et al. 2017. On continuous deployment maturity in customer projects. In Proceedings of the Symposium on Applied Computing (SAC '17). Association for Computing Machinery, New York, NY, USA, 1205–1212. DOI: https://doi.org/10.1145/3019612.3019777	Boa (46%-65%)

EP_20	2016	ACM	Rossi, Chuck et al. 2016. Continuous deployment of mobile software at facebook (showcase). In Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE 2016). Association for Computing Machinery, New York, NY, USA, 12–23. DOI: https://doi.org/10.1145/2950290.2994157	Boa (46%-65%)
EP_21	2016	ACM	Shahin, Mojtaba; Ali Babar, Muhammad; Zhu, Liming. 2016. The Intersection of Continuous Deployment and Architecting Process: Practitioners' Perspectives. In Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM '16). Association for Computing Machinery, New York, NY, USA, Article 44, 1–10. DOI: https://doi.org/10.1145/2961111.2962587	Muito Boa (66%-85%)
EP_22	2016	ACM	Hilton, Michael et al. 2016. Usage, costs, and benefits of continuous integration in open-source projects. In Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering (ASE 2016). Association for Computing Machinery, New York, NY, USA, 426–437. DOI: https://doi.org/10.1145/2970276.2970358	Muito Boa (66%-85%)
EP_23	2019	ACM	Bezemer, Cor-Paul et al. 2019. How is Performance Addressed in DevOps? In Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering (ICPE '19). Association for Computing Machinery, New York, NY, USA, 45–50. DOI: https://doi.org/10.1145/3297663.3309672	Boa (46%-65%)
EP_24	2019	ACM	Kula, Elvan et al. 2019. Releasing fast and slow: an exploratory case study at ING. In Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2019). Association for Computing Machinery, New York, NY, USA, 785–795. DOI: https://doi.org/10.1145/3338906.3338978	Muito Boa (66%-85%)

EP_25	2016	ACM	Laukkanen, Eero et al. 2016. Bottom-up Adoption of Continuous Delivery in a Stage-Gate Managed Software Organization. In Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM '16). Association for Computing Machinery, New York, NY, USA, Article 45, 1–10. DOI:https://doi.org/10.1145/2961111.2962608	Muito Boa (66%-85%)
EP_26	2018	ACM	Humble, Jez. 2018. Continuous delivery sounds great, but will it work here? Commun. ACM 61, 4 (April 2018), 34–39. DOI:https://doi.org/10.1145/3173553	Média (26%-45%)
EP_27	2017	ACM	Zhao, Yangyang et al. 2017. The impact of continuous integration on other software development practices: a large-scale empirical study. In Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE 2017). IEEE Press, 60–71.	Muito Boa (66%-85%)
EP_28	2019	Scopus	Lwakatare, L.E. et al. DevOps in practice: A multiple case study of five companies. Information and Software Technology, vol. 114, pp. 217-230, 2019. doi: 10.1016/j.infsof.2019.06.010	Excelente (>86%)
EP_29	2019	Scopus	Shahin, M.; Zahedi, M.; Babar, M.A.; Zhu, L. An empirical study of architecting for continuous delivery and deployment. Empirical Software Engineering, vol. 24, issue 3, pp. 1061-1108, 2019. doi: 10.1007/s10664-018-9651-4	Excelente (>86%)
EP_30	2019	Scopus	Mårtensson, T.; Ståhl, D.; Bosch, J. Test activities in the continuous integration and delivery pipeline. Journal of Software: Evolution and Process, vol. 31, issue 4, art. no. e2153, 2019. doi: 10.1002/smr.2153	Boa (46%-65%)
EP_31	2018	Scopus	Staron, M.; Meding, W.; Söder, O.; Bäck, M. Measurement and Impact Factors of Speed of Reviews and Integration in Continuous Software Engineering. Foundations of Computing and Decision Sciences, vol 43, issue 4, pp. 281-303, 2018. DOI: 10.1515/fcds-2018-0015	Muito Boa (66%-85%)

EP_32	2018	Scopus	Laukkanen, E. et al. Comparison of release engineering practices in a large mature company and a startup. Empirical Software Engineering, vol. 23, issue 6, pp. 3535-3577, 2018. DOI: 10.1007/s10664-018-9616-7	Muito Boa (66%-85%)
EP_33	2018	Scopus	Pinto, G.; Castor, F.; Bonifacio, R.; Rebouças, M. Work practices and challenges in continuous integration: A survey with Travis CI users. Software - Practice and Experience, vol. 48, issue 12, pp. 2223-2236, 2018. DOI: 10.1002/spe.2637	Muito Boa (66%-85%)
EP_34	2018	Scopus	Rahman, A.; Agrawal, A.; Krishna, R.; Sobran, A. Characterizing the influence of continuous integration: Empirical results from 250+ open source and proprietary projects. SWAN 2018 - Proceedings of the 4th ACM SIGSOFT International Workshop on Software Analytics, co-located with FSE 2018, pp. 8-14, 2018. DOI: 10.1145/3278142.3278149	Média (26%-45%)
EP_35	2018	Scopus	Debroy, V.; Miller, S.; Brimble, L. Building lean continuous integration and delivery pipelines by applying devops principles: A case study at varidesk. ESEC/FSE 2018 - Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp. 851-856, 2018. DOI: 10.1145/3236024.3275528	Média (26%-45%)
EP_36	2018	Scopus	Zhang, Y.; Vasilescu, B.; Wang, H.; Filkov, V. One size does not fit all: An empirical study of containerized continuous deployment workflows. ESEC/FSE 2018 - Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp. 295-306, 2018. DOI: 10.1145/3236024.3236033	Boa (46%-65%)

EP_37	2018	Scopus	Bernardo, J.H.; Da Costa, D.A.; Kulesza, U. Studying the impact of adopting continuous integration on the delivery time of pull requests. Proceedings - International Conference on Software Engineering, pp. 131-141, 2018. DOI: 10.1145/3196398.3196421	Muito Boa (66%-85%)
EP_38	2018	Scopus	Siqueira, R. et al. Continuous delivery: Building trust in a large-scale, complex government organization. IEEE Software, vol. 35, issue 2, pp. 38-43, 2018. DOI: 10.1109/MS.2018.111095426	Muito Boa (66%-85%)
EP_39	2018	Scopus	Ivanov, V.; Smolander, K. Implementation of a DevOps pipeline for serverless applications. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 11271 LNCS, pp. 48-64, 2018. DOI: 10.1007/978-3-030-03673-7_4	Muito Boa (66%-85%)
EP_40	2018	Scopus	Mascheroni, M.A.; Irrazábal, E. Continuous testing and solutions for testing problems in continuous delivery: A systematic literature review. Computacion y Sistemas, 22 (3), pp. 1009-1038, 2018. DOI: 10.13053/CyS-22-3-2794	Excelente (>86%)
EP_41	2018	Scopus	Qumer Gill, A.; Loumish, A.; Riyat, I.; Han, S. DevOps for information management systems. VINE Journal of Information and Knowledge Management Systems, vol. 48, issue 1, pp. 122-139, 2018. DOI: 10.1108/VJIKMS-02-2017-0007	Muito Boa (66%-85%)
EP_42	2017	Scopus	Chen, L. Continuous Delivery: Overcoming adoption challenges. Journal of Systems and Software, vol. 128, pp. 72-86, 2017. DOI: 10.1016/j.jss.2017.02.013	Muito Boa (66%-85%)
EP_43	2017	Scopus	Laukkanen, E.; Itkonen, J.; Lassenius, C. Problems, causes and solutions when adopting continuous delivery — A systematic literature review. Information and Software Technology, vol. 82, pp. 55-79, 2017. DOI: 10.1016/j.infsof.2016.10.001	Muito Boa (66%-85%)

EP_44	2017	Scopus	Da Silva, A. C. B. G. et al. Agility and quality attributes in open source software projects release practices. Proceedings - 2016 10th International Conference on the Quality of Information and Communications Technology, QUATIC 2016, art. no. 7814526, pp. 107-112, 2017. DOI: 10.1109/QUATIC.2016.029	Boa (46%-65%)
EP_45	2017	Scopus	Fitzgerald, B.; Stol, K.-J. Continuous software engineering: A roadmap and agenda. Journal of Systems and Software, vol. 123, pp. 176-189, 2017. DOI: 10.1016/j.jss.2015.06.063	Muito Boa (66%-85%)
EP_46	2017	Scopus	Elberzhager, F. et al. From agile development to devops: Going towards faster releases at high quality - Experiences from an industrial context. Lecture Notes in Business Information Processing, vol. 269, pp. 33-44, 2017. DOI: 10.1007/978-3-319-49421-0_3	Boa (46%-65%)
EP_47	2017	Scopus	Rodríguez, P. et al. Continuous deployment of software intensive products and services: A systematic mapping study. Journal of Systems and Software, vol. 123, pp. 263-291, 2017. DOI: 10.1016/j.jss.2015.12.015	Muito Boa (66%-85%)
EP_48	2016	Scopus	Bae, J.; Kim, C.; Kim, J. Automated deployment of SmartX IoT-cloud services based on continuous integration. 2016 International Conference on Information and Communication Technology Convergence, ICTC 2016, art. no. 7763372, pp. 1076-1081, 2016. DOI: 10.1109/ICTC.2016.7763372	Baixa (<26%)
EP_49	2016	Scopus	Itkonen, J.; Udd, R.; Lassenius, C.; Lehtonen, T. Perceived Benefits of Adopting Continuous Delivery Practices. International Symposium on Empirical Software Engineering and Measurement, art. no. a42, 2016. DOI: 10.1145/2961111.2962627	Média (26%-45%)
EP_50	2016	Scopus	Savor, T. et al. Continuous deployment at Facebook and OANDA. Proceedings - International Conference on Software Engineering, pp. 21-30, 2016. DOI: 10.1145/2889160.2889223	Boa (46%-65%)

EP_51	2016	Scopus	Yaman, S.G. et al. Customer involvement in continuous deployment: A systematic literature review. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 9619, pp. 249-265, 2016. DOI: 10.1007/978-3-319-30282-9_18	Boa (46%-65%)
EP_52	2016	Scopus	Riungu-Kalliosaari, L. et al. DevOps adoption benefits and challenges in practice: A case study. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 10027 LNCS, pp. 590-597, 2016. DOI: 10.1007/978-3-319-49094-6_44	Boa (46%-65%)
EP_53	2017	Scopus	Ullah, F. et al. Security support in continuous deployment pipeline. ENASE 2017 - Proceedings of the 12th International Conference on Evaluation of Novel Approaches to Software Engineering, pp. 57-68, 2017. DOI: 10.5220/0006318200570068	Média (26%-45%)
EP_54	2016	Scopus	Seth, N.; Khare, R. ACI (automated Continuous Integration) using Jenkins: Key for successful embedded Software development (2016) 2015 2nd International Conference on Recent Advances in Engineering and Computational Sciences, RAECS 2015, art. no. 7453279	Média (26%-45%)

APÊNDICE B – PROTOCOLO DA REVISÃO SISTEMÁTICA

HISTÓRICO DE REVISÕES

Revisão	Data	Descrição	Autor
01	11/07/2019	Criação do documento	Rennan Lima
02	05/08/2019	Escrita do Protocolo	Rennan Lima
03	20/08/2019	Escrita do Protocolo	Rennan Lima
04	27/08/2019	Escrita do Protocolo	Rennan Lima
05	30/08/2019	Escrita do Protocolo	Rennan Lima
06	03/02/2020	Revisão do Protocolo	Rennan Lima
07	24/09/2020	Escrita e Revisão	Rennan Lima

EQUIPE

Nome	Organização	Papel
Rennan Francisco Messias de Lima	Cin-UFPE	Autor
Vinicius Cardoso Garcia	Cin-UFPE	Co-Autor e Orientador

1. INTRODUÇÃO

Revisões sistemáticas proveem meios para executar revisões na literatura abrangentes e não tendenciosas, fazendo com que seus resultados tenham valor científico conforme mencionado por Travassos e Biolchini (2007). As revisões sistemáticas têm por objetivo apresentar uma justa avaliação de um tópico de investigação, usando uma confiável, rigorosa e auditável metodologia (KITCHENHAM, 2007).

As vantagens desse tipo de estudo em relação às revisões de literatura tradicionais são: resultados imparciais, mais abrangentes, de maior qualidade e com maior rigor científico. Os motivos para realizar uma SLR estão listados abaixo:

- Sumarizar evidências existentes sobre um fenômeno;
- Identificar lacunas na pesquisa atual;
- Fornecer um arcabouço para posicionar novas pesquisas;
- Apoiar a geração de novas hipóteses.

Uma vez que a metodologia SLR deve ser bem definida, ela deve começar com a definição do protocolo da pesquisa. Este documento contém as questões de pesquisa e a

estratégia que deve ser seguida pelos pesquisadores. Segundo Kitchenham (2007), além das razões e objetivos da pesquisa, devem faze r parte do protocolo:

- As questões de investigação que a pesquisa pretende responder;
- As estratégias usadas para as pesquisas dos estudos primários, incluindo os termos usados, bibliotecas digitais, jornais e conferências;
- Critérios de inclusão e exclusão dos estudos primários;
- Procedimentos de avaliação da qualidade dos estudos selecionados;
- Estratégia de extração dos dados e síntese dos dados extraídos; e,
- Estratégia de documentação e apresentação.

Assim, este documento apresenta o protocolo de uma revisão sistemática, parte de uma pesquisa de mestrado cujo objetivo principal é investigar o que muda no desenvolvimento de software quando são implantadas práticas de integração/entrega contínua. Este estudo busca reunir artefatos que apoiem a adoção dessas práticas no desenvolvimento de software.

2. QUESTÕES DE PESQUISA

Como forma de atualizar as revisões do Laukkanen *et al.* (2017), do Shahin *et al.* (2017) e do Proulx *et al.* (2018), no intuito de encontrar e analisar o maior número de trabalhos primários relevantes e reconhecidos na área que pudessem responder as questões de pesquisa.

Com o objetivo "o que muda no desenvolvimento de software quando são implantadas práticas de integração/entrega contínua?" e "como apoiar o uso dessas práticas?" a pesquisa parte para quatro questões de investigação mais específicas que possam responder essas perguntas na busca por uma abordagem que apoie com práticas e ferramentas eficazes a implantação de integração e entrega contínuas.

Q1. Quais desafios foram relatados para a adoção de práticas contínuas?
Q2. Que práticas foram relatadas para implementar com êxito práticas contínuas?
Q3. Quais ferramentas foram empregadas para projetar e implementar pipelines de implantação?

Kitchenham (2007) recomenda considerar as questões de pesquisa a partir da seguinte estrutura PICOC (*Population*, *Intervention*, *Context*, *Outcomes* e *Comparison*) que traduzida para o português seria: População, Intervenção, Contexto, Resultados e Comparação. Para cada pergunta da pesquisa, são apresentados, a seguir, os elementos PIO (*Population*, *Interventaion* e *Outcome*):

Q1:

- **População** (P): Software ou desenvolvimento de software
- **Intervenção** (I): Adoção de práticas contínuas
- **Resultado** (O): Desafios

Q2:

- **População** (P): Software ou desenvolvimento de software
- **Intervenção** (I): Práticas
- **Resultado** (O): Implementar com êxito práticas contínuas

Q3:

- **População** (P): Software ou desenvolvimento de software
- **Intervenção** (I): Ferramentas
- **Resultado** (O): Projetar e implementar pipelines de implantação

A Comparação e o Contexto da estrutura PICOC não foram utilizados, uma vez que os objetivos do trabalho não incluem nenhum contexto específico e não buscam a comparação entre os tópicos investigados.

3. ESTRATÉGIA DE BUSCA

Segundo Kitchenham (2007), uma estratégia deve ser usada para a pesquisa dos estudos primários, com a definição das palavras chaves, bibliotecas digitais, jornais e conferências. A estratégia usada nessa pesquisa é apresentada nas próximas subseções.

3.1. TERMOS CHAVES DA PESQUISA

A partir das estruturas das questões de investigação (PIO) definidas anteriormente, os principais termos são identificados. Após a identificação, é realizada a tradução desses termos para o inglês por ser a língua utilizada nas bases de dados eletrônicas pesquisadas e nas principais conferências e jornais dos tópicos de investigação.

Além disso, sinônimos são identificados com a orientação de um especialista no tema de investigação para cada um dos principais termos. Como recomendação, os termos chaves identificados serão pesquisados no singular e no plural, para essa variação, foi usado o caractere asterisco (*) que é aceito em muitas bibliotecas digitais e permite a variação de palavras que estejam referenciadas com o símbolo.

Integração contínua: Continuous integration, Rapid integration, Fast integration, Quick integration, Continuous build, Rapid build, Fast build, Quick build;

Entrega contínua: Continuous delivery, Rapid delivery, Fast delivery, Quick delivery;

Implantação contínua: Continuous deployment, Rapid deployment, Fast deployment, Quick deployment, Continuous release, Rapid release, Fast release, Quick release;

Práticas contínua: Continuous practice;

Engenharia contínua: Continuous software engineering, Continuous engineering;

Software: Software*, Program*, System*, Application*, Product*.

3.2. STRING DE BUSCA

As strings de busca são geradas a partir das estruturas das questões e da combinação dos termos chave e sinônimos. São usados OR (ou) entre os sinônimos identificados e AND (e) entre os termos chaves. Algumas adaptações são necessárias de acordo com as necessidades especificas de cada base de dados. Devido a esta revisão tratar-se de uma atualização, o resultado da execução da *string* foi filtrado por publicações a partir de 2016. Possíveis peculiaridades das bibliotecas digitais e adaptações mediante a isso são registradas. As *strings* utilizadas para as questões são listadas a seguir (Quadro 1):

Quadro 1 - String de busca da pesquisa

String da Pesquisa
(“*continuous integration*” *OR* “*rapid integration*” *OR* “*fast integration*” *OR* “*quick integration*” *OR* “*continuous delivery*” *OR* “*rapid delivery*” *OR* “*fast delivery*” *OR* “*quick delivery*” *OR* “*continuous deployment*” *OR* “*rapid deployment*” *OR* “*fast deployment*” *OR* “*quick deployment*” *OR* “*continuous release*” *OR* “*rapid release*” *OR* “*fast release*” *OR* “*quick release*” *OR* “*continuous build*” *OR* “*rapid build*” *OR* “*fast build*” *OR* “*quick build*” *OR* “*continuous practice*” *OR* “*continuous practices*” *OR* “*continuous software engineering*” *OR* “*continuous engineering*”) *AND* (*software** *OR program** *OR system** *OR application** *OR product**)

3.3. FONTES DE BUSCA

Os critérios para a seleção das fontes foram: (1) disponibilidade de consultar os artigos na web; (2) presença de mecanismos de busca usando palavras-chave; e, (3) importância e

relevância das fontes. As fontes de pesquisa utilizadas para a busca dos estudos primários são listadas abaixo:

- *IEEEXplore Digital Library* (https://ieeexplore.ieee.org/)
- *ACM Digital Library* (https://portal.acm.org)
- *Elsevier Scopus* (https://www.scopus.com/)
- *Elsevier ScienceDirect* (www.sciencedirect.com)

Outras fontes foram inicialmente consideradas como potenciais para as buscas: *Google*, *Google Scholar*, *SpringerLink*, *ISI Web of Science* e *Wiley Online Library*. Entretanto, estas foram posteriormente excluídas da lista final de fontes por algumas das seguintes razões:

- Algumas por não estarem presentes em importantes revisões sistemáticas ou não terem sido recomendadas por especialistas;
- Algumas por não permitirem a visualização ou download dos trabalhos sem pagamento ou licenças que a instituição de realização do trabalho não possuía;
- Algumas por já ser indexadas por algumas das fontes já listadas na pesquisa.

4. SELEÇÃO DOS ESTUDOS

Os estudos que podem fazer parte dessa pesquisa são:

- Artigos em Jornais, Revistas, Conferências e Congressos;
- Relatórios Técnicos;
- Dissertações e Teses;

Além disso, outros estudos não previstos que sejam encontrados e possam contribuir para a pesquisa, podem ser adicionados. Caso isso aconteça, ou qualquer outra mudança no processo de busca, será relatada na seção 6 (Documentação do Processo de Busca).

Uma vez que estudos potencialmente candidatos a se tornarem estudos primários tenham sido obtidos, eles precisam ser analisados para que a sua relevância seja confirmada e trabalhos com pouca relevância sejam descartados. Segundo Travassos e Biolchini (2007) critérios de inclusão e exclusão devem ser baseados nas questões de pesquisa. Logo, alguns critérios de inclusão e exclusão são definidos nas próximas subseções, baseados nos trabalhos de Kitchenham (2007) e Travassos e Biolchini (2007).

4.1. CRITÉRIOS DE INCLUSÃO

A inclusão de um trabalho é determinada pela relevância em relação às questões de investigação, determinada pela análise do título, palavras-chave, resumo, introdução e conclusão. Os seguintes critérios de inclusão foram definidos:

a) Estudos que tratem primariamente ou secundariamente de desafios na adoção de práticas contínuas;
b) Estudos que tratem primariamente ou secundariamente Boas Práticas, Lições Aprendidas e Fatores de Sucesso relacionados à Integração ou Entrega Contínua;
c) Estudos que tratem primariamente ou secundariamente de ferramentas associadas para facilitar as práticas contínuas.

4.2. CRITÉRIOS DE EXCLUSÃO

A partir também da análise do título, palavras-chave, resumo, introdução e conclusão, são excluídos os estudos que se enquadrem em qualquer dos casos abaixo:

a) Estudos que não respondam nenhuma das questões de pesquisa;
b) Estudos que não estejam disponíveis livremente para consulta na web ou Portal da Capes;
c) Estudos que apresentem texto, conteúdo e resultados incompletos, ou seja, trabalhos com resultados não concluídos não serão aceitos;
d) Estudos claramente irrelevantes para a pesquisa, de acordo com as questões de investigação levantadas;
e) O foco principal do artigo ser avaliar uma nova tecnologia ou ferramenta em um caso da vida real;
f) Não são artigos científicos revisados por pares (por exemplo, apresentações, chamadas para trabalhos, discursos, prefácios etc.) ou capítulos de livros e livros;
g) Estudos Repetidos: se determinado estudo estiver disponível em diferentes fontes de busca, a primeira pesquisa será considerada;
h) Estudos Duplicados: caso dois trabalhos apresentem estudos semelhantes, apenas o mais recente e/ou o mais completo será incluído, a menos que tenham informação complementar;
i) Não estar em português ou inglês;

j) *Short papers* (menos de 6 páginas).

5. PROCESSO DE SELEÇÃO DOS ESTUDOS PRIMÁRIOS

De acordo com Kitchenham (2007), as buscas iniciais retornam uma grande quantidade de estudos que não são relevantes, não respondendo às questões ou mesmo não tendo relação com o tópico em questão. Então, estudos totalmente irrelevantes são descartados no início. A seguir, são apresentadas as etapas do processo de seleção dos estudos primários.

- O pesquisador inicialmente realiza as buscas para identificar os potenciais estudos primários e a partir da leitura dos títulos dos trabalhos que a pesquisa retorna e palavra-chave, excluem trabalhos que claramente são irrelevantes para as questões investigadas;
- A partir da lista com os potenciais candidatos a estudos primários, todos os trabalhos são avaliados, mediante a leitura do resumo e conclusão, considerando-se os critérios de inclusão e exclusão, para então se chegar a uma lista final de estudos primários;
- Os estudos incluídos são documentados através de formulários, assim como todos os trabalhos excluídos e o critério que definiu sua exclusão. Posteriormente, cada estudo primário é lido e através de formulários a extração dos dados e avaliação da qualidade dos trabalhos é realizada.

Formulário A

O formulário A deverá ser utilizado para armazenar dados relativos aos trabalhos incluídos no estudo.

Trabalhos Incluídos						
Id	**Fonte**	**Título**	**Autor**	**Local de publicação**	**Tipo**	**Ano**

Formulário B

O formulário B deverá ser utilizado para armazenar dados relativos aos trabalhos não incluídos no estudo

Trabalhos Excluídos							
Id	**Fonte**	**Título**	**Autor**	**Local de publicação**	**Tipo**	**Ano**	**Critério usado para exclusão**

Formulário C

O formulário C deverá ser utilizado para extração dos dados relativos aos trabalhos incluídos no estudo.

FORMULÁRIO DE COLETA DE DADOS		
ID:	**Pesquisador:**	**Data da Avaliação:**
Título do Trabalho:		
Autores:		
Fonte de Pesquisa:	**Tipo:**	
Local da Publicação:		**Ano:**
Tipo de Estudo:		
Domínio da Aplicação	**Contexto:**	
[x] INCLUIDO - Critérios Utilizados:		
QUESTÕES DE PESQUISA		
Q1: Quais desafios foram relatados para a adoção de práticas contínuas?		
Q2: Que práticas foram relatadas para implementar com êxito práticas contínuas?		
Q3: Quais ferramentas foram empregadas para projetar e implementar pipelines de implantação?		

AVALIAÇÃO DA QUALIDADE		
Item	**Critérios de Qualidade**	**Valores**
Introdução/Planejamento		
1	Os objetivos ou questões do estudo são claramente definidos (incluindo justificativas para a realização do estudo)?	
2	O tipo de estudo está definido claramente?	
Desenvolvimento		
3	Existe uma clara descrição do contexto no qual a pesquisa foi realizada?	
4	O trabalho é bem/adequadamente referenciado (apresenta trabalhos relacionados/semelhantes e baseia-se em modelos e teorias da literatura)?	
Conclusão		
5	O estudo relata de forma clara e não ambígua os resultados?	
6	Os objetivos ou questões do estudo são alcançados?	
Critério Específico para estudos Experimentais		
7	Existe um método ou um conjunto de métodos descrito para a realização do estudo?	
Critério Específico para estudos Teóricos		
7	Existe um processo não tendencioso na escolha dos estudos?	
Critério Específico para Revisões Sistemáticas		
7	Existe um protocolo rigoroso, descrito e seguido?	
Critério Específico para Relato de Experiência Industrial		
7	Existe uma descrição sobre a(s) organização(ões), equipe(s), projeto(s) e distribuição envolvida?	
Critérios para as Questões de Investigação (Q1, Q2 e Q3)		
8	O estudo lista primária ou secundariamente de desafios na adoção de práticas contínuas?	
9	O estudo lista primária ou secundariamente Boas Práticas, Lições Aprendidas e Fatores de Sucesso relacionados à Integração ou Entrega Contínua?	
10	O estudo lista primária ou secundariamente de ferramentas associadas para facilitar as práticas continuas?	
TOTAL		
Observações/Comentários:		

6. DOCUMENTAÇÃO DO PROCESSO DE BUSCA

O processo de execução de revisão sistemática deve ser transparente e replicável. Assim, toda a revisão, bem como a busca devem ser documentadas conforme forem executadas e mudanças devem ser anotadas e justificadas. Tendo como base as diretrizes de Kitchenham (2007), essa seção aborda as limitações e adaptações que ocorram no processo de busca definido para essa revisão.

A primeira limitação encontrada e prevista é quanto às fontes de busca, algumas ainda não estão preparadas para este tipo de abordagem e em outras a sintaxe das *strings* de busca precisam sofrer adaptações, bem como, as bases terem limitação quanto ao tamanho da *string*, fazendo com que o pesquisador tenha que dividir a *string* em várias partes para poder executar, com isso o número de artigos repetidos aumenta. Como o número de trabalhos retornados era grande, o trabalho tornou-se demorado.

Com isso, para auxiliar a execução da revisão decidiu-se utilizar a ferramenta StArt[1] que permite a importação dos artigos através do formato BibTex, que em algumas bases há limitação quanto a quantidade de artigos a podem ser exportados em lote, e que tem como objetivo auxiliar o pesquisador, dando suporte à aplicação da Revisão Sistemática.

Com o uso da ferramenta StArt, a exclusão dos artigos repetidos se dá de forma automática, o que reduziu o tempo de trabalho nessa tarefa. O foco do pesquisador primeiramente passou a ser excluir trabalhos claramente irrelevantes para a pesquisa através da análise do título, resumo e palavra-chave.

Após essa primeira seleção, os estudos foram analisados através da introdução e conclusão, onde os trabalhos podem ser excluídos ou incluídos como estudos primários da pesquisa.

7. AVALIAÇÃO DA QUALIDADE DOS ESTUDOS

Em adição aos critérios gerais de inclusão e exclusão, é considerado importante avaliar a qualidade dos estudos primários (KITCHENHAM, 2004). Apesar de não existir uma definição universal do que seja qualidade de estudo, a maioria dos checklists incluem questões que objetivam avaliar a extensão em que o viés é minimizado e a validação interna e externa são maximizadas (KHAN et al., 2001; KITCHENHAM, 2007).

[1] http://lapes.dc.ufscar.br/tools/start_tool/

7.1. TIPO DE ESTUDO

Os tipos de estudos são classificados conforme Easterbrook, Singer *et al.* (2007):

- Experimentais ou *Empirical Studies*;
- Teóricos (estudos conceituais baseados em um entendimento de uma área, referenciando outros trabalhos relacionados);
- Revisões Sistemáticas (estudos secundários, onde os trabalhos são reexaminados).
- Relato de Experiência Industrial (*Industrial Experience Report*).

Os métodos para estudos experimentais (conjunto de princípios de organização em torno do qual os dados empíricos são coletados e analisados) são classificados por Easterbrook, Singer *et al.* (2008) como: Experimentos controlados, Estudos de caso, Estudo de Campo, Etnografia e Pesquisa-Ação.

Um experimento controlado é uma investigação de uma hipótese testável, uma pré-condição para a realização de um experimento é uma hipótese clara. A hipótese (teoria a partir da qual o experimento é desenhado) guia todas as etapas do projeto experimental, incluindo a de decidir quais as variáveis a incluir no estudo e como medi-las (EASTERBROOK, SINGER, *et al.*, 2008).

Os estudos de caso oferecem uma compreensão profunda de como e porque certos fenômenos ocorrem. Estudos de caso exploratórios são usados como investigações iniciais de alguns fenômenos para derivar novas hipóteses e construir teorias. Já os Estudos de caso de confirmação são usados para testar as teorias existentes (EASTERBROOK, SINGER, *et al.*, 2008).

Um Estudo de Capo é usado para identificar as características de uma ampla população de indivíduos. É mais estreitamente associado com o uso de questionários para coleta de dados. No entanto, um Estudo de Capo também pode ser realizado por meio de entrevistas estruturadas, técnicas ou registro de dados (EASTERBROOK, SINGER, *et al.*, 2008).

Para a engenharia de software, a etnografia pode ajudar a compreender como as comunidades técnicas constroem uma cultura de práticas e estratégias de comunicação que lhes permitem executar os trabalhos técnicos de forma colaborativa (EASTERBROOK, SINGER, *et al.*, 2008).

Em uma Pesquisa-Ação, os investigadores tentam resolver um problema do mundo real e simultaneamente estudar a experiência de resolver o problema (DAVISON, MARTINSONS e KOCK, 2004).

7.2. CRITÉRIOS DE AVALIAÇÃO

Para a realização da avaliação dos estudos primários, algumas questões são definidas, essas questões estão disponíveis nessa seção e fazem parte do formulário C da Seção 5. Dentre os critérios de avaliação, existem alguns que deverão ser aplicados a todos os tipos de estudo e outros que são específicos para cada tipo de estudo (Experimental, Teórico, Revisões Sistemáticas e Relatos de Experiência Industrial).

Além de perguntas relacionadas à como o estudo foi conduzido e os resultados de cada trabalho avaliado, foram adicionadas 3 (três) perguntas relacionadas às questões de investigação, no intuito de verificar o quanto cada estudo atende aos objetivos desta pesquisa. Como as questões são semelhantes e complementares, um mesmo trabalho pode apresentar resultados para as 4 (quatros) questões de pesquisa e assim obter bons conceitos nos três critérios.

Para a avaliação da qualidade dos estudos é usada a escala Likert-5, que permite respostas gradativas. Para responder as questões dos critérios de qualidade. O pesquisador pode usar os seguintes níveis de concordância ou discordância (concordo totalmente, concordo parcialmente, neutro, discordo parcialmente e discordo totalmente). O Quadro 2 apresenta as questões da avaliação da qualidade dos estudos. Para a avaliação, devem ser consideradas as seguintes observações:

Concordo totalmente (4): deve ser concedido no caso em que o trabalho apresente no texto os critérios que atendam totalmente a questão;

Concordo parcialmente (3): deve ser concedido no caso em que o trabalho atenda parcialmente aos critérios da questão;

Neutro (2): deve ser concedido no caso em que o trabalho não deixe claro se atende ou não a questão;

Discordo parcialmente (1): deve ser concedido no caso em que os critérios contidos na questão não são atendidos pelo trabalho avaliado;

Discordo totalmente (0): deve ser concedido no caso em que não existe nada no trabalho que atenda aos critérios da questão.

Quadro 2 - Questões para a Avaliação da Qualidade dos Estudos

Item	Critérios de Qualidade	Valores
Introdução/Planejamento		
1	Os objetivos ou questões do estudo são claramente definidos (incluindo justificativas para a realização do estudo)?	
2	O tipo de estudo está definido claramente?	
Desenvolvimento		
3	Existe uma clara descrição do contexto no qual a pesquisa foi realizada?	
4	O trabalho é bem/adequadamente referenciado (apresenta trabalhos relacionados/semelhantes e baseia-se em modelos e teorias da literatura)?	
Conclusão		
5	O estudo relata de forma clara e não ambígua os resultados?	
6	Os objetivos ou questões do estudo são alcançados?	
Critério Específico para estudos Experimentais		
7	Existe um método ou um conjunto de métodos descrito para a realização do estudo?	
Critério Específico para estudos Teóricos		
7	Existe um processo não tendencioso na escolha dos estudos?	
Critério Específico para Revisões Sistemáticas		
7	Existe um protocolo rigoroso, descrito e seguido?	
Critério Específico para Relato de Experiência Industrial		
7	Existe uma descrição sobre a(s) organização(ões), equipe(s), projeto(s) e distribuição envolvida?	
Critérios para as Questões de Investigação (Q1, Q2 e Q3)		
8	O estudo lista primária ou secundariamente de desafios na adoção de práticas contínuas?	
9	O estudo lista primária ou secundariamente Boas Práticas, Lições Aprendidas e Fatores de Sucesso relacionados à Integração ou Entrega Contínua?	
10	O estudo lista primária ou secundariamente de ferramentas associadas para facilitar as práticas continuas?	
TOTAL		

8. ESTRATÉGIA DE EXTRAÇÃO DOS DADOS

Para apoiar a extração e registro dos dados e posterior análise, foi utilizada a ferramenta StArt, que tem como objetivo auxiliar o pesquisador, dando suporte à aplicação da Revisão Sistemática. A ferramenta foi bastante útil pois possibilita a importação no formato BibTex dos resultados das pesquisas nas bases de dados, o cadastro do protocolo, e dos formulários de extração de dados e de avaliação da qualidade dos estudos, permitindo que o pesquisador registre o motivo que levou a exclusão do estudo. Cada alteração feita é salva e

ao final da pesquisa o arquivo serve como documentação da revisão e pode ser usado por outro pesquisador a fim de repeti-la.

Na ferramenta, para cada trabalho aprovado pelo processo de seleção, o pesquisador registrava o critério de inclusão, ao final foi extraída uma listagem com os trabalhos incluídos, com apenas as informações que identificam o trabalho e dados que serão apresentados em forma de gráficos nos resultados da revisão. Para os trabalhos excluídos foi registrado na ferramenta o motivo que levou a exclusão. O formulário extrair as informações gerais e realização da avaliação da qualidade, foi cadastrado e preenchido na ferramenta.

9. SÍNTESE DOS DADOS COLETADOS

Após a coleta dos dados, as informações devem ser tabuladas de acordo com as questões de pesquisa, as tabelas devem ser estruturadas de forma a destacar as semelhanças e diferenças entre os resultados do estudo (KITCHENHAM, 2007; TRAVASSOS e BIOLCHINI, 2007). Kitchenham (2007) afirma que a síntese dos dados pode ser quantitativa e/ou qualitativa, sendo que a primeira necessariamente seria tratada através de meta-análise. Nesta pesquisa, a natureza dos dados é qualitativa, logo uma síntese qualitativa é realizada.

Os dados extraídos dos estudos são organizados em tabelas exportados da ferramenta StArt para uma planilha eletrônica. A partir de similaridades dos dados extraídos, utilizando-se para isso o método de comparações constantes, é realizada a síntese dos dados e são listados desafios, boas práticas e ferramentas identificadas na adoção da integração, entrega e implantação contínua para responder a cada questão de pesquisa.

O processo se inicia com a marcação de trechos dos textos dos trabalhos (ou dados qualitativos) que fornecem informação relevante para responder as questões de pesquisa. A cada um desses trechos são associados a códigos de cores que indicam que tipo, ou categoria, de informação o trecho está provendo. O procedimento de análise desses dados extraídos e sintetizados é apresentado na próxima subseção.

10. DOCUMENTAÇÃO E APRESENTAÇÃO DOS RESULTADOS

A fase final de uma revisão sistemática envolve a redação dos resultados da análise e divulgação dos resultados aos potenciais interessados. Alguns estudos indicam alguns tópicos necessários para a apresentação de uma revisão sistemática: Título (de acordo com as questões de pesquisa); Autores; Resumo do trabalho (contexto, objetivos, método, resultado s e conclusões); Background (justificativa da necessidade da revisão); Questões da pesquisa;

Método da revisão (estratégia de busca, seleção dos estudos, avaliação da qualidade, extração e síntese dos dados); Estudos incluídos e excluídos; Resultados; Discussão, e Conclusões (KITCHENHAM, 2007; TRAVASSOS e BIOLCHINI, 2007).

A partir da síntese dos dados, um conjunto de desafios e melhores práticas para adoção de integração, entrega e implantação contínua serão identificados e documentados, além da apresentação de ferramentas de apoio. Junto à exposição dos resultados dessa pesquisa, este protocolo deve ser disponibilizado.

REFERÊNCIAS

DAVISON, R. M.; MARTINSONS, M. G.; KOCK, N. Principles of Canonical Action Research. **Information Systems Journal, vol. 14, issue 1**, 2004. 65-86.

EASTERBROOK, S. et al. Selecting Empirical Methods for Software Engineering Research. In: SHULL, F., et al. **Guide to Advanced Empirical Software Engineering**. London: Springer, 2008. p. 285-311.

KITCHENHAM, B. Procedures for Performing Systematic Reviews. **Joint Technical Report, Software Engineering Group, Keele University**, 2004.

KITCHENHAM, B. **Guidelines for performing Systematic Literature Reviews in Software Engineering**. V 2.3, EBSE Technical Report. Durham. 2007.

LAUKKANEN, E.; ITKONEN, J.; LASSENIUS, C. Problems, causes and solutions when adopting continuous delivery - A systematic literature review. **Information and Software Technology**, v. 82, p. 55-79, 1 fevereiro 2017.

PROULX, A. et al. Problems and Solutions of Continuous Deployment: A Systematic Review, 2018.

SHAHIN, M.; ALI BABAR, M.; ZHU, L. Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices. **IEEE Access, 5**, 2017. 3909-3943.

TRAVASSOS, G.; BIOLCHINI, J. **Revisões Sistemáticas Aplicadas a Engenharia de Software**. XXI SBES - Brazilian Symposium on Software Engineering. João Pessoa: SBES 2007 - XXI Simpósio Brasileiro de Engenharia de Software. 2007.

APÊNDICE C – DESAFIOS DE APOIO A ADOÇÃO DO PIPELINE CI/CD

Categoria	D: Desafios na adoção de pipeline de implantação	Referências – EP: Estudos Primários	Quantidade de Trabalhos - Qualidade
Humano e Organizacional	D1. Cliente desmotivado	EP_14	1 (1 Excelente)
	D2. Curva de aprendizado íngreme	EP_28, EP_36	2 (1 Excelente e 1 Boa)
	D3. Falta de habilidades	EP_19, EP_28	2 (1 Excelente e 1 Boa)
	D4. Desafios organizacionais	EP_25, EP_42	2 (2 Muito Boa)
	D5. Falta de comunicação	EP_07, EP_33, EP_52	3 (2 Muito Boa e 1 Boa)
	D6. Dependências da equipe	EP_21	1 (1 Muito Boa)
	D7. Falsa sensação de confiança	EP_33	1 (1 Muito Boa)
	D8. Falta de conhecimento de domínio	EP_33	1 (1 Muito Boa)
	D9. Questões Culturais	EP_19, EP_26, EP_52	3 (2 Boa e 1 Média)
Processo	D10. Ambientes heterogêneos	EP_21, EP_52	2 (1 Muito Boa e 1 Boa)
	D11. Múltiplos Ambientes	EP_33	1 (Muito Boa)
Ferramentas	D12. Ferramentas em constante mudança	EP_21	1 (1 Muito Boa)
	D13. Falta de suporte de ferramenta	EP_35	1 (1 Média)

Design de Compilação	D14. Longos tempos de espera para builds	EP_35, EP_36	2 (1 Boa e 1 Média)
Integração	D15. Múltiplos Branches	EP_05, EP_25	2 (1 Excelente e 1 Muito Boa)
	D16. Falhas de Build	EP_04, Ep_25, EP_33	3 (3 Muito Boa)
	D17. Branches longas	EP_03, EP_07	2 (1 Muito Boa e 1 Boa)
Arquitetura da Aplicação	D18. Dependências	EP_14, EP_29, EP_33	3 (2 Excelente e 1 Muito Boa)
	D19. Arquitetura Altamente Acoplada	EP_21, EP_25, EP_26, EP_29	4 (1 Excelente, 2 Muito Boa e 1 Média)
	D20. Restrições de Domínio	EP_14	1 (1 Excelente)
Teste	D21. Baixa cobertura de teste	EP_14, EP_25	2 (1 Excelente e 1 Muito Boa)
	D22. Falta de teste de aceitação automatizado	EP_14	1 (1 Excelente)
	D23. Testes de Longa Duração	EP_14	1 (1 Excelente)
	D24. Falta de testes	EP_17, EP_32, EP_33	3 (3 Muito Boa)
	D25. Testes instáveis	EP_25, EP_33	2 (2 Muito Boa)
	D26. Falta de estratégia de teste	EP_25	1 (1 Muito Boa)
	D27. Teste Inadequado	EP_33	1 (Muito Boa)
Entrega	D28. Falta de Mecanismo de Reversão	EP_14	1 (1 Excelente)
	D29. Processo de Entrega Demorado	EP_03, EP_35	2 (1 Boa e 1 Média)

Infraestrutura	D30. Falta de recursos	EP_19, EP_28, EP_32, EP_33, EP_35	5 (1 Excelente, 2 Muito Boa, 1 Média e 1 Boa)
	D31. Configuração manual de software	EP_14	1 (1 Excelente)
	D32. Esforço Inicial	EP_05	1 (1 Excelente)

APÊNDICE D – MELHORES PRÁTICAS DE APOIO A ADOÇÃO DO PIPELINE CI/CD

MP: Melhores práticas de apoio à adoção de pipeline de implantação	**Referências – EP: Estudos Primários**	**Quantidade de Trabalhos - Qualidade**
MP1. Adoção de Práticas Ágeis	EP_06, EP_25, EP_28, EP_42, EP_45, EP_49	6 (1 Excelente, 4 Muito Boa e 1 Média)
MP2. Padronizar o Fluxo de trabalho no Sistema de Controle de Versões	EP_28, EP_39	2 (1 Excelente e 1 Muito Boa)
MP3. Mudanças no esquema do banco de dados no sistema de controle de versão	EP_39	1 (1 Muito Boa)
MP4. Todos os commits estão ligados às tarefas	EP_39	1 (1 Muito Boa)
MP5. Commit de código com mais frequência	EP_27, EP_31, EP_34, EP_37	4 (3 Muito Boa e 1 Média)
MP6. Commits pequenos	EP_27, EP_31, EP_34, EP_37	4 (3 Muito Boa e 1 Média)
MP7. Trabalhar em Pequenos lotes	EP_15	1 (1 Muito Boa)
MP8. Refatoração	EP_06	1 (1 Muito Boa)
MP9. Revisão de Código	EP_06, EP_28, EP_32, EP_39 EP_50	5 (1 Excelente, 3 Muito Boa e 1 Boa)
MP10. Análise Estática de Código Automática	EP_04, EP_06, EP_14, EP_20, EP_28, EP_31, EP_39, EP_49	8 (2 Excelente, 4 Muito Boa, 1 Boa e 1 Média)
MP11. Coleta de métricas de qualidade	EP_06	1 (1 Muito Boa)

MP12. Testes automatizados	EP_27, EP_28, EP_31, EP_46, EP_48, EP_49	6 (1 Excelente, 3 Muito Boa, 1 Média e 1 Baixa)
MP13. Definir Estratégias de testes	EP_06	1 (1 Muito Boa)
MP14. Melhorar a atividade de teste	▪ TDD [EP_06, EP_08, EP_42] ▪ BDD [EP_06, EP_42] ▪ Testes de segurança [EP_14, EP_15, EP_39, EP_42] ▪ Testes unitários [EP_20, EP_21, EP_25, EP_28, EP_30, EP_31, EP_32, EP_36, EP_38, EP_39, EP_42, EP_50] ▪ Testes de integração [EP_20, EP_21, EP_23, EP_25, EP_26, EP_28, EP_32, EP_36, EP_38, EP_42, EP_50] ▪ Teste de cobertura [EP_17, EP_30, EP_32, EP_39] ▪ Smoke testes [EP_15, EP_32] ▪ Build teste [EP_20] ▪ Teste de regressão [EP_16, EP_20, EP_22, EP_26] ▪ Teste de sistema [EP_30]	22 (3 Excelente, 11 Muito Boa, 6 Boa, 1 Média e 1 Baixa)

	▪ Testes de aceitação [EP_15, EP_16, EP_21, EP_25, EP_26, EP_28, EP_38, EP_42] ▪ Teste funcional [EP_29, EP_31] ▪ Teste de recuperação de desastre [EP_15] ▪ Snapshot testes [EP_20] ▪ Teste de perfomance [EP_20, EP_23, EP_29, EP_39, EP_42, EP_48, EP_50] ▪ Teste de capacidade [EP_20] ▪ Testes de conformidade [EP_20] ▪ Testes A/B [EP_16]	
MP15. Paralelização de Testes automatizados	EP_14, EP_32	2 (1 Excelente e 1 Muito Boa)
MP16. Microsserviços	EP_13, EP_17, EP_21, EP_26, EP_28, EP_29, EP_35	7 (2 Excelente, 3 Muito Boa e 2 Média)
MP17. Build automatizado	EP_28, EP_31, EP_39, EP_48	4 (1 Excelente, 2 Muito Boa e 1 Baixa)
MP18. Implantação automatizada	EP_28, EP_48	3 (1 Excelente e 1 Baixa)

MP19. Definindo uma estratégia de implantação	▪ Feature Flags [EP_13] ▪ Implantação Canário [EP_13, EP_28] ▪ Implantação Dark Launches [EP_13, EP_50] ▪ Implantação Blue-Green [EP_28, EP_50] ▪ Rolling Upgrade [EP_28] ▪ Implantação Staging/Baking [EP_50]	3 (1 Excelente, 1 Muito Boa e 1 Boa)
MP20. Definir Mecanismo de Rollback	EP_28	1 (1 Excelente)

APÊNDICE E – FERRAMENTAS DE APOIO A ADOÇÃO DO PIPELINE CI/CD

Categoria	F: Ferramentas de apoio a adoção de pipeline de implantação	Referências – EP: Estudos Primários	Quantidade de Trabalhos - Qualidade
Gerenciamento de projeto	F1. IBM Rational Team Concert (RTC)	EP_05	1 (1 Excelente)
	F2. Jira	EP_12	1 (1 Muito Boa)
	F3. Atlassian Confluence	EP_15	1 (1 Muito Boa)
Controle de Versão	F4. GitHub	EP_04, EP_22, EP_27, EP_28, EP_33, EP_34, EP_36, EP_37, EP_48, EP_53	10 (1 Excelente, 5 Muito Boa, 1 Boa, 2 Média e 1 Baixa)
	F5. Git	EP_02, EP_06, EP_15, EP_23, EP_28, EP_31, EP_54	7 (1 Excelente, 3 Muito Boa, 1 Média, 1 Boa e 1 Baixa)
	F6. BitBucket	EP_12, EP_28	2 (1 Excelente e 1 Muito Boa)
	F7. Deveo	EP_28	1 (1 Excelente)
	F8. GitLab	EP_11, EP_18, EP_24, EP_38, EP_39	5 (3 Muito Boa e 2 Boa)
	F9. Apache Subversion (SVN)	EP_23	1 (1 Boa)
	F10. Azure DevOps	EP_35	1 (1 Média)

Análise estática de código	F11. SonarQube	EP_05, EP_06, EP_10, EP_24, EP_28	5 (2 Excelente, 2 Muito Boa e 1 Boa)
	F12. CodeSonar	EP_12	1 (1 Muito Boa)
	F13. Gerrit	EP_31	1 (1 Muito Boa)
	F14. Klocwork	EP_12	1 (1 Muito Boa)
	F15. Attack Surface Analyzer	EP_10	1 (1 Boa)
	F16. Microsoft Baseline Security Analyzer	EP_10	1 (1 Boa)
Build	F17. Maven	EP_04, EP_05, EP_06, EP_24, EP_49	5 (1 Excelente, 3 Muito Boa e 1 Média)
	F18. Gradle	EP_04, EP_06	2 (2 Muito Boa)
	F19. Ant	EP_06	1 (1 Muito Boa)
Servidor CI	F20. Jenkins	EP_02, EP_05, EP_06, EP_09, EP_10, EP_11, EP_18, EP_23, EP_24, EP_28, EP_31, EP_34, EP_46, EP_48, EP_49, EP_53, EP_54	17 (3 Excelente, 3 Muito Boa, 4 Média, 5 Boa e 2 Baixa)
	F21. Travis CI	EP_04, EP_09, EP_22, EP_27, EP_33, EP_34, EP_36, EP_37	8 (6 Muito Boa, 1 Média e 1 Boa)
	F22. CircleCI	EP_22, EP_34, EP_36	3 (1 Muito Boa, 1 Boa e 1 Média)
	F23. GitLab CI/CD	EP_11, EP_39	2 (1 Muito Boa e 1 Boa)
	F24. AppVeyor	EP_22	1 (1 Muito Boa)

	F25. Bamboo CI	EP_12	1 (1 Muito Boa)
	F26. CloudBees	EP_22	1 (1 Muito Boa)
	F27. GoCD	EP_09	1 (1 Muito Boa)
	F28. TeamCity	EP_15	1 (1 Muito Boa)
	F29. Wercker	EP_22	1 (1 Muito Boa)
	F30. Azure DevOps	EP_35	1 (1 Média)
Testes	F31. IBM Rational Quality Manager (RQM)	EP_05	1 (1 Excelente)
	F32. Selenium	EP_28	1 (1 Excelente)
	F33. Cucumber	EP_15	1 (1 Muito Boa)
	F34. Fortify	EP_24	1 (1 Muito Boa)
	F35. Gatling	EP_15	1 (1 Muito Boa)
	F36. JMeter	EP_10	1 (1 Boa)
	F37. Compatibility Test Suite by Android (CTS)	EP_54	1 (1 Média)
	F38. JUnit	EP_53	1 (1 Média)
	F39. Smokemonster	EP_49	1 (1 Média)
Repositório de artefatos	F40. Artifactory	EP_06, EP_24	2 (2 Muito Boa)
	F41. Nexus	EP_02	1 (1 Baixa)
Configuração e	F42. AWS Cloud	EP_09, EP_28, EP_39	3 (1 Excelente e 2 Muito Boa)
	F43. Puppet	EP_15, EP_18, EP_23, EP_28	4 (1 Excelente, 1 Muito Boa e 2 Boa)

	F44. Vagrant	EP_12, EP_28, EP_49	3 (1 Excelente, 1 Muito Boa e 1 Média)
	F45. Chef	EP_28, EP_38	2 (1 Excelente e 1 Muito Boa)
	F46. Docker	EP_28, EP_35, EP_36, EP_46, EP_48	5 (1 Excelente, 2 Boa, 1 Média e 1 Baixa)
	F47. Ansible	EP_02, EP_28, EP_49	3 (1 Excelente, 1 Média e 1 Baixa)
	F48. Plataforma System Center Orchestrator	EP_05	1 (1 Excelente)
	F49. Chake	EP_38	1 (1 Muito Boa)
	F50. Nolio	EP_24	1 (1 Muito Boa)
	F51. SSH (deploy)	EP_23	1 (1 Boa)
	F52. Kubernetes	EP_35	1 (1 Média)
Monitoramento	F53. New Relic	EP_23, EP_28	2 (1 Excelente e 1 Boa)
	F54. AWS Monitoring Services	EP_28	1 (1 Excelente)
	F55. CA Service Operations Insight (SOI)	EP_05	1 (1 Excelente)
	F56. GrayLog	EP_28	1 (1 Excelente)
	F57. Kinesis	EP_28	1 (1 Excelente)
	F58. Apache Kafka	EP_06	1 (1 Muito Boa)
	F59. Linux Tracing Toolkit Next Generation	EP_12	1 (1 Muito Boa)

F60. Nagios	EP_02, EP_23	2 (1 Boa e 1 Baixa)
F61. Dynatrace	EP_23	1 (1 Boa)
F62. System	EP_23	1 (1 Boa)
F63. Dataloop.io	EP_49	1 (1 Média)

APÊNDICE F – JENKINSFILE

```
#!groovy

pipeline {
    agent none
    stages {
        stage('Checkout'){
            agent any
            steps{
                checkout scm
                sh 'python --version'
                sh 'git log HEAD^..HEAD --pretty="%h %an - %s" > GIT_CHANGES'
            }
        }
        stage('Install Application Dependencies') {
            agent { dockerfile true }
            steps {
                echo '=========== Install Application Dependencies ============'
                sh 'python -m venv .venv'
                sh 'pwd'
                sh '. .venv/bin/activate'
                sh '.venv/bin/pip install -U pip'
                sh '.venv/bin/pip install -r requirements-test.txt'
                sh 'cp contrib/env-sample-test .env'
                sh '.venv/bin/python manage.py compilemessages'
            }
        }
        stage('Run Unit/Integration Tests'){
            agent { dockerfile true }
            steps {
                echo "=========== Run Unit/Integration Tests ============"
                sh '.venv/bin/coverage run manage.py test'
                sh '.venv/bin/coverage report'
                sh '.venv/bin/coverage html'
                sh '.venv/bin/coverage xml'
            }
            post {
                always {
                    cobertura autoUpdateHealth: false, autoUpdateStability: false,
                    ↪ coberturaReportFile: 'reports/code-coverage/coverage.xml',
                    ↪ conditionalCoverageTargets: '70, 0, 0', failNoReports: false, failUnhealthy:
                    ↪ false, failUnstable: false, lineCoverageTargets: '80, 0, 0',
                    ↪ maxNumberOfBuilds: 10, methodCoverageTargets: '80, 0, 0', onlyStable: false,
                    ↪ zoomCoverageChart: false
```

```
39                  publishHTML([allowMissing: false, alwaysLinkToLastBuild: false, keepAll: false,
                    ↪  reportDir: 'htmlcov/', reportFiles: 'index.html', reportName: 'Coverage
                    ↪  Report HTML', reportTitles: ''])
40                  junit allowEmptyResults: true, testResults: 'reports/test-reports/*.xml'
41              }
42          }
43      }
44      stage('Static Analysis') {
45          parallel {
46              stage('Lint Test'){
47                  agent { dockerfile true }
48                  steps{
49                      sh '.venv/bin/pycodestyle manhanah'
50                  }
51                  post {
52                      always {
53                          recordIssues(tools: [pep8(pattern: '**/pep8.report')])
54                      }
55                  }
56              }
57              stage('SonarQube Analysis') {
58                  agent any
59                  environment {
60                      scannerHome = tool name: 'sonar_scanner', type:
                        ↪  'hudson.plugins.sonar.SonarRunnerInstallation'
61                      MANHANAH_SONAR_TOKEN = credentials('manhanah-project-token')
62                  }
63                  steps {
64                      echo '=========== SonarQube analysis ============'
65                      withSonarQubeEnv('SonarQube') {
66                          sh "${scannerHome}/sonar-scanner-4.5.0.2216-linux/bin/sonar-scanner -e \
67                                  -Dsonar.host.url=http://sonarqube.ifac.edu.br \
68                                  -Dsonar.projectVersion=${env.BUILD_NUMBER} \
69                                  -Dsonar.login=${MANHANAH_SONAR_TOKEN}"
70                      }
71                  }
72              }
73          }
74      }
75      stage('Quality Gate'){
76          agent any
77          steps {
78              timeout(time: 1, unit: 'HOURS') {
79                  // Parameter indicates whether to set pipeline to UNSTABLE if Quality Gate fails
80                  // true = set pipeline to UNSTABLE, false = don't
81                  waitForQualityGate abortPipeline: true
```

```
82                 }
83             }
84
85         }
86         stage('Deploy to Staging'){
87             agent { dockerfile true }
88             environment {
89                 STAGING_SSH_CREDS = credentials('staging-ssh-creds')
90             }
91             steps {
92                 sh """
93                     .venv/bin/fab deploy --stage-name=staging --username=${STAGING_SSH_CREDS_USR}
↪  --key-filename=${STAGING_SSH_CREDS}
94                 """
95             }
96         }
97         stage('Run User Acceptance Tests'){
98             agent { dockerfile true }
99             steps {
100                echo "=========== Run User Acceptance Tests ============"
101                sh '.venv/bin/python manage.py behave'
102            }
103        }
104        stage('Run Security Tests'){
105            agent { dockerfile true }
106            steps {
107                echo '=========== Deployment checklist ============'
108                sh 'rm -rf .env'
109                sh 'cp contrib/env-sample .env'
110                sh '.venv/bin/python manage.py check --deploy --fail-level ERROR'
111            }
112        }
113    }
114    post{
115        always {
116            node(null) {
117                echo 'One way or another, I have finished'
118                deleteDir() /* clean up our workspace */
119            }
120        }
121        failure{
122            mail body: "Build failed :face_with_head_bandage:
↪  \n'${env.JOB_NAME}#${env.BUILD_NUMBER}' <${env.BUILD_URL}|Open in Jenkins>",
↪  subject: '[MANHANAH] Build failed Jenkins', to: 'cosis@ifac.edu.br'
123        }
124    }
```

```
125 }
```

APÊNDICE G – GITLAB-CI.YML

```yaml
default:
  image: python:3.7

variables:
  PROJECT_SONAR_TOKEN: $MANHANAH_SONAR_TOKEN
  SONARQUBE_URL: http://sonarqube.ifac.edu.br

workflow:
  rules:
    # Executar apenas pipelines para solicitações de mesclagem, tags e para a branch principal
    - if: '$CI_PIPELINE_SOURCE == "merge_request_event"'
    - if: '$CI_COMMIT_BRANCH == $CI_DEFAULT_BRANCH'
    - if: $CI_COMMIT_TAG

stages:
  - build
  - test
  - qa
  - security scan
  - deploy staging
  - functional tests
  - deploy production

.before_script_template:
  before_script:
    - python -V
    - apt-get update && apt-get upgrade -y
    - apt-get install -y libsasl2-dev python-dev libldap2-dev libssl-dev libpq-dev postgresql-client
↪  gettext
    - pip install -U pip setuptools wheel
    - pip install --no-cache-dir -U -r requirements-test.txt
    - cp contrib/env-sample-test .env

include:
  - template: Security/Dependency-Scanning.gitlab-ci.yml
  - template: Security/License-Scanning.gitlab-ci.yml
  - template: Security/SAST.gitlab-ci.yml
  - template: Security/Secret-Detection.gitlab-ci.yml

compile_messages:
  extends: .before_script_template
  stage: build
  script:
```

```
    - python manage.py compilemessages

migrations:
  extends: .before_script_template
  stage: build
  services:
    - name: postgres:9.6
      alias: db-postgres
  variables:
    POSTGRES_DB: manhanah_ci
    POSTGRES_USER: manhanahciuser
    POSTGRES_PASSWORD: $POSTGRES_PASSWORD
    POSTGRES_HOST_AUTH_METHOD: trust
    DATABASE_URL: "postgresql://$POSTGRES_USER:$POSTGRES_PASSWORD@db-postgres:5432/$POSTGRES_DB"
  script:
    - python manage.py check
    - python manage.py migrate

django-tests:
  extends: .before_script_template
  stage: test
  script:
    - coverage run manage.py test
    - coverage report
    - coverage html
    - coverage xml
  coverage: '/TOTAL.*\s([.\d]+)%/'
  artifacts:
    when: always
    paths:
      - htmlcov
      - reports
    reports:
      coverage_report:
        coverage_format: cobertura
        path: reports/code-coverage/coverage.xml
      junit: reports/test-reports/*.xml

lint-tests:
  extends: .before_script_template
  stage: qa
  script:
    - isort --diff -c manhanah
    - pycodestyle manhanah

sonarqube-check:
```

```yaml
  image: sonarsource/sonar-scanner-cli:latest
  stage: qa
  script:
    - pwd
    - ls
    - /usr/bin/entrypoint.sh sonar-scanner -e -Dsonar.host.url=$SONARQUBE_URL
↪  -Dsonar.projectVersion=$CI_PIPELINE_IID -Dsonar.login=$PROJECT_SONAR_TOKEN
↪  -Dsonar.qualitygate.wait=true
  needs:
    - job: django-tests
      artifacts: true

dependencies-check:
  extends: .before_script_template
  stage: security scan
  script:
    - echo "Check Dependencies Security Vulnerabilities"
    - safety check --ignore=41002 --full-report

secret_detection:
  stage: security scan

dependency_scanning:
  stage: security scan

license_scanning:
  stage: security scan

sast:
  stage: security scan
  variables:
    VALIDATE_SCHEMA: "true"

deploy-check:
  extends: .before_script_template
  stage: security scan
  script:
    - echo "Deployment checklist"
    - rm -rf .env
    - cp contrib/env-sample .env
    - python manage.py check --deploy --fail-level ERROR
  only:
    - master
    - tags

deploy-staging:
```

```
133     stage: deploy staging
134     environment:
135       name: staging
136       url: https://www2.ifac.edu.br/sisrad
137     before_script:
138       - 'command -v ssh-agent >/dev/null || ( apt-get update -y && apt-get install openssh-client -y )'
139       - eval $(ssh-agent -s)
140       - echo "$SSH_PRIVATE_KEY_STAGING" | tr -d '\r' | ssh-add -
141       - mkdir -p ~/.ssh
142       - chmod 700 ~/.ssh
143       - ssh-keyscan $VM_IPADDRESS_STAGING >> ~/.ssh/known_hosts
144       - chmod 644 ~/.ssh/known_hosts
145       - pip install fabric
146     script:
147       - fab deploy --stage-name=staging --build-number=$CI_PIPELINE_IID
148     only:
149       - master
150       - tags
151
152   acceptance-tests:
153     extends: .before_script_template
154     stage: functional tests
155     script:
156       - python manage.py behave
157     only:
158       - master
159       - tags
160     needs:
161       - job: deploy-staging
162
163
164   deploy-production:
165     stage: deploy production
166     environment:
167       name: production
168       url: https://rad.ifac.edu.br
169     before_script:
170       - 'command -v ssh-agent >/dev/null || ( apt-get update -y && apt-get install openssh-client -y )'
171       - eval $(ssh-agent -s)
172       - echo "$SSH_PRIVATE_KEY_PRODUCTION" | tr -d '\r' | ssh-add -
173       - mkdir -p ~/.ssh
174       - chmod 700 ~/.ssh
175       - ssh-keyscan $VM_IPADDRESS_PRODUCTION >> ~/.ssh/known_hosts
176       - chmod 644 ~/.ssh/known_hosts
177       - pip install fabric
178     script:
```

```
    - fab deploy --stage-name=production --tag=$CI_COMMIT_TAG --build-number=$CI_PIPELINE_IID
  only:
    - tags
  needs:
    - job: acceptance-tests

notify-sentry-deploy:
  image: getsentry/sentry-cli
  stage: deploy production
  variables:
    SENTRY_AUTH_TOKEN: $SENTRY_AUTH_TOKEN
    SENTRY_PROJECT: $SENTRY_PROJECT
    SENTRY_ORG: $SENTRY_ORG
    SENTRY_URL: $SENTRY_URL
    SENTRY_ENVIRONMENT: production
  script:
    - export SENTRY_URL=$SENTRY_URL
    - export SENTRY_RELEASE=$SENTRY_PROJECT@$(echo $CI_COMMIT_TAG | sed 's/v//g')
    - sentry-cli releases new -p $SENTRY_PROJECT $SENTRY_RELEASE
    - sentry-cli releases set-commits $SENTRY_RELEASE --auto
    - sentry-cli releases finalize $SENTRY_RELEASE
    - sentry-cli releases deploys $SENTRY_RELEASE new -e $SENTRY_ENVIRONMENT
  only:
    - tags
  needs:
    - job: deploy-production
```