



Universidade Federal de Pernambuco
Centro de Informática

Bacharelado em Engenharia da Computação

Pesquisa sobre práticas de Integração e Deployment contínuos em Recife

Mateus Valgueiro Teixeira

Trabalho de Graduação

Recife
05 de maio de 2021

Universidade Federal de Pernambuco
Centro de Informática

Mateus Valgueiro Teixeira

**Pesquisa sobre práticas de Integração e Deployment
contínuos em Recife**

Trabalho apresentado ao Programa de Bacharelado em Engenharia da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Bacharel em Engenharia da Computação.

Orientador: *Paulo Henrique Monteiro Borba*
Co-orientadora: *Klissiomara Lopes Dias*

Recife
05 de maio de 2021

Agradecimentos

Gostaria de agradecer a meus pais e minha irmã por todo apoio desde o começo da minha vida até o presente momento. Agradeço também a Heitor Samuel Carvalho Souza por ter me ajudado durante a fase de entrevistas e por ter cedido todo o material de base deste trabalho. Por fim, gostaria de agradecer ao Professor Paulo Borba e Professora Klissiomara Lopes pela orientação e discussões construtivas durante o desenvolvimento texto.

Só sei que nada sei.
—SÓCRATES

Resumo

As práticas de *Continuous Integration* [CI], *Continuous Delivery* e *Continuous Deployment* [CD] já estão presentes no cotidiano de grandes empresas de todo o mundo. E isto é devido, entre outras coisas, aos comprovados benefícios que a utilização destas técnicas trazem para a equipe e para o produto em desenvolvimento. Mesmo assim, poucos são os estudos que investigam o estado da arte destas práticas na maioria das empresas, principalmente no contexto da cidade de Recife. Com isso, o presente trabalho objetiva entender como as técnicas de integração, entrega e implantação contínuas foram importadas para as empresas recifenses, assim como identificar princípios e práticas adjacentes que governam a adoção destas técnicas. Para tanto, foi realizada uma pesquisa qualitativa por meio de entrevistas com 11 desenvolvedores de software de empresas de tecnologia sediadas na cidade do Recife. Foi descoberto que a amostra não segue o *Stairway to Heaven*, teoria definida pelo artigo base deste trabalho e que sugere que a adoção deve seguir uma sequência específica de práticas. Não obstante, a maioria entrevistada integra o código de uma nova funcionalidade para a *branch* principal apenas no final da *Sprint*, não diariamente, o que não é consistente com as definições mais comuns de CI. Além disso, as práticas de entregas parciais são utilizadas apenas por uma pequena parcela de desenvolvedores.

Palavras-chave: Integração Contínua, Deployment Contínuo, Desenvolvimento Colaborativo, Engenharia de Software

Abstract

< vou adicionar depois da revisão do resumo>

Keywords: DIGITE AS PALAVRAS-CHAVE AQUI

Sumário

1	Introdução	1
1.1	Estrutura do Trabalho	1
2	Fundamentação Teórica	3
2.1	Metodologias Ágeis	3
2.1.1	SCRUM	4
2.2	DevOps	5
2.2.1	Integração Contínua	6
2.2.2	Entrega e <i>Deployment</i> Contínuos	6
2.2.3	Entregas Parciais	7
2.3	Code Review	8
2.4	Trabalhos Relacionados	8
3	Motivação	9
3.1	A grande adoção de CI e CD	9
3.2	A falta de estudos no contexto Recifense	9
3.3	O artigo base	10
3.4	Perguntas de pesquisa	11
4	Metodologia	12
4.1	Pré-estudo	12
4.2	Estrutura da Entrevista	12
4.3	Participantes	13
4.4	Análise dos Dados	14
5	Resultados	18
5.1	Estudo de Predomínio das Práticas	18
5.2	Stairway to heaven	19
5.3	Análise das Práticas	20
5.3.1	Integração Contínua	21
5.3.1.1	Trunk Based Development [TBD]	21
5.3.1.2	Feature Toggles [FT]	21
5.3.1.3	Developer Awareness [AWA]	22
5.3.2	Deployment Contínuo	23
5.3.2.1	Health Checks [HC]	23
5.3.2.2	Developer on Call [DOC]	23

5.3.2.3	Deployment Pipeline [PIP]	24
5.3.3	Entregas Parciais	24
5.3.3.1	Canary Releases [CAN]	24
5.3.3.2	Dark Launches [DAR]	24
5.3.3.3	Testes A/B [AB]	25
5.4	Descobertas adicionais	25
5.4.1	Code Review	25
5.4.2	Testes Automáticos	26
5.5	Ameaças a validade	26
6	Conclusão	27
6.1	Trabalhos Futuros	27
A	Questionário Traduzido	28
A.1	Demografia	28
A.2	Processo de entrega em geral	28
A.3	Papéis/Responsabilidades	29
A.4	Garantia da Qualidade	29
A.5	Gerenciamento de Problemas	30
A.6	Avaliação da entrega	31
A.7	Finalização	31
B	Tabela Entrevistado-Pratica-Codigos	32
C	Super Categorias	37

Lista de Figuras

2.1	O processo SCRUM	4
2.2	A cultura DevOps	5
3.1	Stairway to Heaven	10
4.1	Detalhamento dos entrevistados	14
4.2	Distribuição dos Participantes por gênero	14
4.3	Distribuição dos Participantes por tamanho da empresa	14
4.4	Fluxograma da Metodologia	15
4.5	Trecho da Tabela Entrevistado-Pratica-Codigos	16
4.6	Exemplo de super categoria	17
5.1	Tabela 1 do artigo base	20

Lista de Tabelas

5.1	Nível de utilização das práticas, com as colunas em ordem decrescente de uso	19
5.2	Diferença entre a ordem de predomínio das práticas	21
5.3	Nível de utilização das práticas, com as colunas na ordem do <i>Stairway to Heaven</i>	22

CAPÍTULO 1

Introdução

As práticas de integração, entrega e implantação contínuos [12, 13] (*Continuous Integration* [CI], *Continuous Delivery* e *Continuous Deployment* [CD]) já são muito difundidas e utilizadas por empresas de tecnologia em todo o mundo. Segundo a pesquisa da empresa *digital.ia* [1], 55% dos participantes reportaram que sua organização utiliza a técnica de integração contínua. Também nesta mesma pesquisa foi encontrado que 41% utilizam a técnica de *Continuous Delivery* e 36%, *Continuous Deployment*.

Estes números elevados são causados principalmente pelos benefícios que a utilização destas técnicas trazem para a equipe e para o produto em desenvolvimento. Estudos comprovam que os desenvolvedores envolvidos se sentem mais produtivos quando usam as práticas de CI/CD [16] e o seu uso traz maior qualidade ao software que está sendo produzido [25].

É possível, no entanto, perceber uma grande falta de estudos a respeito de como essas práticas foram importadas para as empresas de todo o mundo [23], incluindo Recife – as pesquisas são geralmente voltadas apenas para as corporações globais como Facebook [25] e Google [21]. A capital pernambucana é um dos grandes pólos tecnológicos do Brasil: somente no Porto Digital, localizado na cidade, há cerca de 330 empresas e 11 mil trabalhadores com faturamento anual de R\$ 2,3 bilhões em 2019 [3].

Um estudo a respeito de como as técnicas de CI/CD migraram para a indústria de Recife serve como ponto de partida para pesquisas relacionadas ao levantamento de dores sentidas pelos desenvolvedores locais que inibem a utilização destas técnicas.

Neste contexto, este trabalho tem como objetivo entender, através de uma pesquisa qualitativa, quais as práticas e técnicas subjacentes de integração, entrega e *deployment* contínuos adentraram nas empresas de Recife. Não obstante, o trabalho deseja descobrir que princípios e práticas subjacentes governam a adoção destas técnicas.

1.1 Estrutura do Trabalho

O trabalho está divido em 6 capítulos, segmentados da seguinte forma:

- O Capítulo 2 contém a Fundamentação Teórica deste trabalho, abordando sobre as práticas de CI/CD e definindo termos que serão utilizados durante o trabalho, além de trabalhos relacionados
- O Capítulo 3 apresenta a motivação por trás deste trabalho
- O Capítulo 4 contém a metodologia utilizada para a montagem e execução das entrevistas, além de todos os passos da análise de dados

- No Capítulo 5 encontram-se os resultados obtidos com base nas entrevistas
- O Capítulo 6 contém a conclusão com uma análise final e perspectivas de trabalhos futuros.

CAPÍTULO 2

Fundamentação Teórica

Este capítulo tem como objetivo apresentar conceitos importantes para a construção desta pesquisa e entendimento dos resultados obtidos. Assim, o capítulo abordará conceitos de metodologias ágeis, DevOps [2], técnicas de integração, entrega e *deployment* contínuos [12, 13] e práticas relacionadas. Por fim, serão abordados alguns dos trabalhos relacionados a este.

2.1 Metodologias Ágeis

A citação a seguir retirada da dissertação [11] define bem como os métodos ágeis surgiram.

“Durante a evolução dos processos de Engenharia de Software, a indústria se baseou nos métodos tradicionais de desenvolvimento de software, que definiram por muitos anos os padrões para criação de software nos meios acadêmico e empresarial. Porém, percebendo que a indústria apresentava um grande número de casos de fracasso, alguns líderes experientes adotaram modos de trabalho que se opunham aos principais conceitos das metodologias tradicionais. Aos poucos, foram percebendo que suas formas de trabalho, apesar de não seguirem os padrões no mercado, eram bastante eficientes. Aplicando-as em vários projetos, elas foram aprimoradas e, em alguns casos, chegaram a se transformar em novas metodologias de desenvolvimento de software. Essas metodologias passaram a ser chamadas de leves por não utilizarem as formalidades que caracterizavam os processos tradicionais e por evitarem a burocracia imposta pela utilização excessiva de documentos. Com o tempo, algumas delas ganharam destaque nos ambientes empresarial e acadêmico, gerando grandes debates, principalmente relacionados à confiabilidade dos processos e à qualidade do software.”

—DAIRTON LUIZ BASSI FILHO

É notório que a antecipação total de requisitos, como propunha o modelo Cascata antigo, era a fórmula perfeita para a falha do projeto, como comenta [15]. Principalmente em ambientes de desenvolvimento de software, a metodologia mais interessante deve se basear em entregas parciais rápidas, com o objetivo de encontrar erros em fases iniciais e gerar um produto final mais de acordo com a expectativa do cliente. A ideia, como comenta o artigo, é reduzir o custo

de mudanças durante o todo o desenvolvimento do projeto. As metodologias ágeis surgiram então com o objetivo de suprir essas necessidades.

No geral, é possível perceber que as metodologias ágeis conseguiram encantar as empresas de desenvolvimento de software principalmente por dar a habilidade de gerenciar mudanças de prioridades, ajudar no alinhamento entre as equipes de mercado e de tecnologia e aumentar a velocidade de entregas, de acordo com [1]. Pesquisas como esta demonstram o sucesso que esse novo paradigma obteve no contexto de produção de aplicações de T.I.

2.1.1 SCRUM

Uma das metodologias ágeis que apresenta uma grande quantidade de usuários é o SCRUM [24]. Ele é um framework de gerenciamento de projetos que estabelece uma série de regras e cerimônias ao processo com o objetivo de garantir entregas iterativas e incrementais. Mesmo tendo sido criado em 1993 por Jeff Suntherland, a metodologia é – entre as ágeis – a mais utilizada atualmente: de acordo com a pesquisa da *digital.ia* de 2020 [1], 58% das equipes utilizam o framework na sua organização.

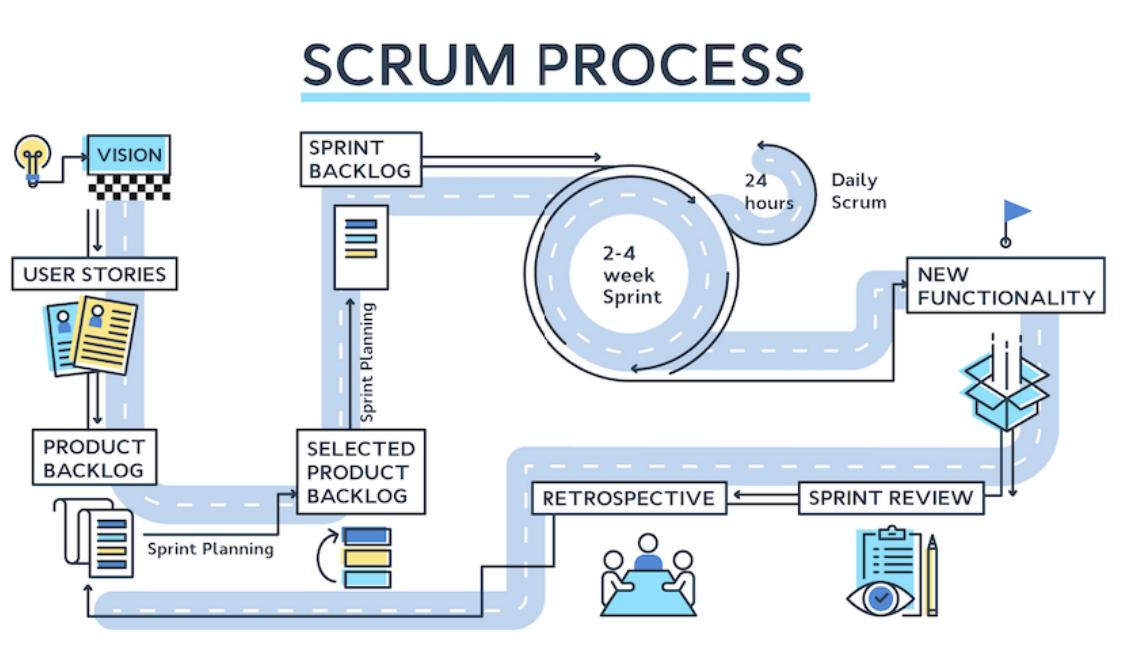


Figura 2.1 O processo SCRUM. Fonte [6]

A Figura 2.1 mostra o processo da metodologia. No SCRUM, os ciclos de cada projeto são denominados *Sprints*, que têm duração geralmente de uma semana e são planejados durante o *Planning*. Esta reunião acontece no início da *Sprint* e serve para decidir o conjunto de tarefas a ser feito pela equipe durante o tempo estipulado. As tarefas são retiradas do *Backlog* do produto, que contém todos os objetivos e funcionalidades já acertados com o cliente que deverão ser feitos pela equipe.

Todos os dias acontece a chamada *Daily meeting*: uma reunião diária para facilitar o acompanhamento do projeto, levantar discussão a respeito das atividades desenvolvidas para disseminar conhecimento e identificar possíveis impedimentos dentro da equipe. Estas reuniões, de acordo com o método, não devem durar mais do que 15 minutos, e, por este motivo, devem ser feitas em pé.

Ao final da *Sprint* acontece a reunião de *Review*, onde a equipe apresenta o que foi realizado durante a última iteração e os resultados do trabalho. Após ela, acontece também a cerimônia principal para garantia de melhoria contínua: a retrospectiva. Nela, a equipe discute melhorias que podem ser implementadas no processo para atingir uma velocidade maior de entregas.

Com a pesquisa da empresa *digital.ia* [1] é possível perceber que, mesmo não aplicando o SCRUM definido por Suntherland, várias das cerimônias definidas na metodologia são utilizadas: 85% dos entrevistados fazem *Daily's*, 81% utilizam a cerimônia de retrospectiva e 77%, *Planning*.

2.2 DevOps

Mesmo com a flexibilidade e agilidade das metodologias ágeis comentadas na seção anterior, as empresas ainda estavam necessitadas de um método de reduzir o tempo entre entregas de novas features. Esta foi a principal motivação para a criação da cultura de *DevOps* [20].

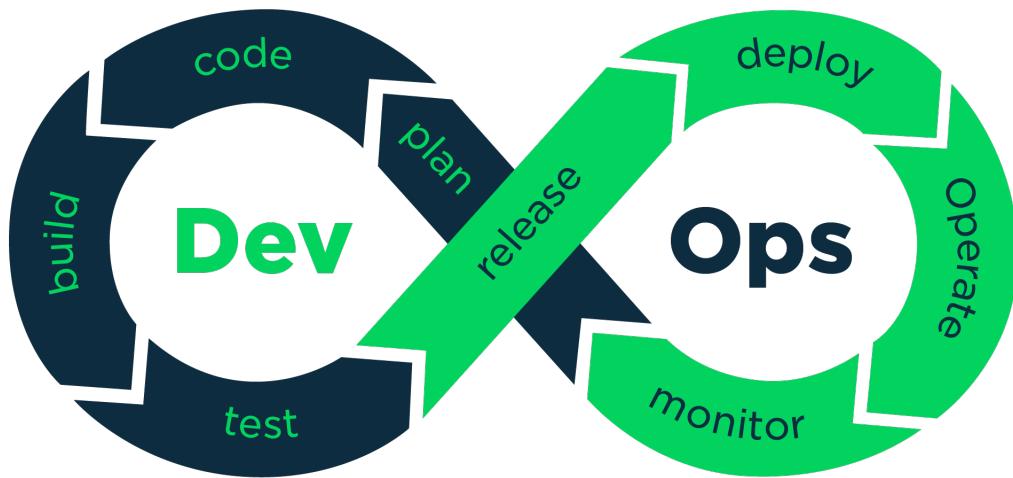


Figura 2.2 A cultura DevOps

DevOps é definido como uma série de práticas que buscam reduzir o tempo entre a finalização de uma nova funcionalidade até que esta esteja em produção, garantindo ainda o máximo de qualidade da mesma. Mesmo contendo o tempo como uma das principais preocupações, é importante também para o método observar a importância da qualidade do que está sendo entregue em vários aspectos: segurança, disponibilidade, confiabilidade. Não adianta entregar

funcionalidades rapidamente se essas não se comportam de acordo com o esperado ou expõem o software a ataques maliciosos.

Para atingir o seu objetivo, de acordo com [2], a cultura de *DevOps* prega a quebra de silos entre os desenvolvedores de software (Dev) e os administradores de sistema (Ops). Melhorar a comunicação entre os dois papéis e impulsionar a automação de processos e o monitoramento em todas as fases de desenvolvimento de um software auxiliam as empresas no gerenciamento de lançamento de novas versões e no processo de garantia de qualidade.

Dentre as práticas sugeridas pelo método, estão: *Continuous Integration* (integração contínua); *Continuous Delivery* (entrega contínua) e *Deployment*; e *Partial Rollouts* (entregas parciais). Estas serão abordadas nas subseções seguintes.

2.2.1 Integração Contínua

A técnica de *Continuous Integration* (integração contínua ou CI), de acordo com [13], têm como principal característica a integração de código no repositório compartilhado múltiplas vezes por dia. Para garantir a qualidade do software que está sendo integrado, cada *merge* passa por testes automáticos pré-programados que garantem que cenários de usabilidade estão de acordo com os requisitos definidos.

Algumas práticas foram definidas dentro da técnica para definir processos chaves que produzem um CI eficiente. Entre elas, temos o *trunk based development* [TBD] [9], onde todo o time contribui para uma única *branch* no repositório de versionamento. Esta técnica é comumente utilizada para diminuir a quantidade de conflitos entre linhas de códigos que estão sendo concorrentemente modificadas por desenvolvedores diferentes.

Para garantir que o *trunk based development* funcione como esperado, deve-se ter uma ferramenta que possibilite o chaveamento de partes de código que ainda não foram finalizadas. A implementação comum desta técnica é *feature toggle* [FT] [17]. A técnica define, através de condicionais adicionadas no código fonte, que blocos devem ser executados ou ignorados no ambiente de produção.

Outra prática relacionada a técnica de integração contínua é o chamado *developer awareness* [AWA] [22], que prega a quebra de silos entre os desenvolvedores e processos de *release*, o status dos ambientes e informações gerais da infraestrutura do sistema que está sendo desenvolvido.

2.2.2 Entrega e *Deployment* Contínuos

A técnica de *Continuous Delivery* (entrega contínua ou CD) define que o software tem que estar a qualquer momento pronto para ser enviado para produção no repositório principal [12]. Para que a técnica de entrega contínua funcione como o esperado, é necessário principalmente que o software esteja pronto para *deploy* durante todo o ciclo de vida e que o time sempre priorize deixar o código pronto para produção ao invés de priorizar novas funcionalidades.

Muitas vezes a técnica de *Continuous Delivery* é confundida com a de *Continuous Deployment*. No entanto, esta última significa que o código é automaticamente colocado em produção sempre que possível, resultando em múltiplos *deployments* por dia. Com a entrega contínua, a empresa pode escolher ter um processo de entregas mais devagar, mesmo tendo o software

sempre pronto para utilização. Outro ponto importante é que para se ter *deployment* contínuo, é necessário ter entrega contínua.

A técnica de *Continuous deployment* foi bastante difundida através do exemplo de empresas como a *Flickr*, que em 2009 comentou sobre seus mais de 10 *deploys* diários [4], antes mesmo o termo *DevOps* ser inventado.

Algumas práticas definem alguns dos passos necessários para utilizar a técnica de CD. Podemos citar o *deployment pipeline* [PIP] [20], que define o conjunto de passos que qualquer mudança de código tem que passar para chegar em produção. Esses passos podem tratar da compilação do código, da execução de testes em diferentes ambientes, entre outras coisas e pode ser totalmente ou parcialmente automatizada.

Depois da entrega de uma nova versão, *health checks* [HC] [20] são necessários para garantir que o produto está funcionando corretamente. O sistema tem uma série de parâmetros definidos pela equipe que mostram se há algum problema no ambiente de produção, por exemplo, falha na conexão com o banco de dados, serviços inativos, certificados HTTPS expirados, entre outros. Muitas vezes o sistema tem a funcionalidade de envio de mensagens para os responsáveis quando algo de errado ocorre no ambiente.

Para garantir ainda mais uma colaboração maior entre desenvolvedores e o time de operações, surgiu a prática de *developer on call* [DOC] [9]. Esta sugere que o desenvolvedor responsável pela funcionalidade recém lançada fique disponível por tempo extra após o lançamento em produção. Caso haja algum erro, ele será a pessoa mais propícia a resolvê-lo o mais rápido possível.

2.2.3 Entregas Parciais

A prática de *partial rollouts* (ou entregas parciais) pode ser definida como um processo de garantia de qualidade e validação de requisitos que ocorrem em tempo de execução [23]. Dentre as técnicas ligadas, podemos citar *canary releases* [CAN] [18], que define o ato de enviar versões apenas para uma parte dos usuários ativos. Isto serve para testar mudanças no software primeiramente com uma parcela pequena de clientes.

Outra prática relacionada a técnica é a de Testes A/B [AB] [19]. Ela se baseia no teste concorrente de duas ou mais versões rodando paralelamente, que se diferem em um ponto isolado do sistema. A ideia é avaliar através de medidas estatísticas de uso e performance do sistema quais das versões é a mais pertinente aos requisitos do software. O teste pode mostrar desde versões que representam um tempo de resposta mais rápido para o usuário, até o lugar que um botão deve aparecer para vender mais produtos.

A prática de *dark launches* [DAR] [9] também se enquadra dentro das técnicas de entregas parciais. Ela é utilizada para testes de funcionalidades no ambiente de produção, mas sem a necessidade de habilitar a mesma para os usuários. Geralmente é utilizado para testar situações encontradas especificamente apenas em produção.

2.3 Code Review

A técnica de *code review* [5] consiste em uma inspeção manual das mudanças por desenvolvedores diferentes daquele responsável pelo desenvolvimento da mudança. Esta é uma técnica utilizada desde projetos Open Source até softwares industriais. Serve principalmente como forma de compartilhar conhecimento e fomentar discussões produtivas a respeito de soluções produzidas pela equipe.

2.4 Trabalhos Relacionados

Há vários trabalhos relacionados a benefícios e barreiras que existem na adoção de integração contínua em contextos variados. É possível perceber que no geral os desenvolvedores gostam de utilizar CI por garantir um desenvolvimento de uma forma mais segura e confiável [21] e por se sentirem mais produtivos [16]. É possível perceber também que os desenvolvedores ainda acham que as ferramentas de CI são complicadas e difíceis de configurar [16].

É interessante levantar ainda no contexto de integração contínua que utilizar as ferramentas voltadas para esta técnica sem adequar a cultura de desenvolvimento para tal leva a práticas não saudáveis de desenvolvimento, tais como *builds* que levam muito tempo para concluir ou permanecem quebrados por longos períodos de tempo [10]. Outro estudo encontrou que a utilização da técnica apresenta um *trade-off* entre velocidade e certeza no que diz respeito a garantia de qualidade do software [16].

Já a respeito de *deployment* contínuo, foi encontrada a visão dos desenvolvedores do Facebook e de OANDA a respeito desta técnica [25]. O artigo mostra que os desenvolvedores preferem *deployments* mais rápidos por gerar maior qualidade de software e maior produtividade. Contudo, eles acreditam que há uma instabilidade maior e é uma metodologia inviável para sistemas críticos.

Podemos ainda citar o artigo mais relacionado e que também serve de base para este trabalho [23], que apresenta uma pesquisa empírica a respeito dos princípios e práticas de entrega e *deployment* contínuos em empresas européias e norte-americanas. Os autores encontram que umas das principais barreiras para a adoção de CD são problemas arquiteturais. Além disso, percebe-se que *feature toggles* como uma implementação de técnicas de entregas parciais levam a uma complexidade de código indesejada.

CAPÍTULO 3

Motivação

Neste capítulo será abordada a motivação por trás da pesquisa produzida. Na primeira seção, será falado um pouco sobre a grande adoção de CI/CD pela indústria. Já na segunda seção, será abordada a falta de estudos voltados para o contexto Recifense. A terceira discorre sobre o artigo base que serviu de motivação para este trabalho. A última seção contém as perguntas de pesquisa que este trabalho tenta responder.

3.1 A grande adoção de CI e CD

As práticas de integração e *deployment* contínuos (*Continuous Integration* e *Continuous Deployment*) já são muito difundidas e utilizadas por empresas de tecnologia em todo o mundo. Segundo a pesquisa realizada pela empresa *digital.ai* [1], 55% dos participantes reportaram que sua organização pratica a técnica de integração contínua. Também nesta mesma pesquisa foi encontrado que 36% utilizam a técnica de *deployment* contínuos.

Estes números elevados são causados principalmente pelos benefícios que a utilização destas técnicas trazem para a equipe e para o produto em desenvolvimento. Ainda de acordo com [1], entre as razões para adoção de CI/CD, as principais são aceleração de entregas de software (71%), e aumentar a produtividade (51%) e a qualidade do software (42%).

Especificamente sobre integração contínua, a prática hoje em dia já é bastante estudada difundida na indústria. O estudo [16] comenta que desenvolvedores envolvidos se sentem mais produtivos quando utilizam a prática e dão mais valor aos testes automáticos. Já com relação a *deployment* contínuo, o estudo [25] mostra que seu uso traz maior qualidade ao software que está sendo produzido.

3.2 A falta de estudos no contexto Recifense

Ainda há, no entanto, uma grande falta de estudos a respeito de como essas práticas foram importadas para a maioria das empresas [23], inclusive as de Recife, um dos grandes polos tecnológicos do Brasil. Somente no Porto Digital, localizado na capital pernambucana, há cerca de 330 empresas e 11 mil trabalhadores com faturamento anual de R\$ 2,3 bilhões em 2019 [3].

O presente trabalho procura entender sobre a utilização das práticas de CI/CD nas empresas de Recife para que esses dados sirvam como objeto de pesquisa para levantamento de possíveis dores sentidas pelos desenvolvedores locais que justifiquem a adoção ou não destas práticas. Um estudo a respeito de como as técnicas de integração e *deployment* contínuo migraram para a

indústria de Recife funciona ainda como um *benchmark* de como as empresas estão se portando em relação às novidades presentes na literatura nos últimos anos, assim como levantar possíveis discrepâncias entre empresas situadas na área e as grandes corporações.

3.3 O artigo base

Com o objetivo de entender um pouco mais sobre o estado da prática de CI/CD no contexto de Recife, nos baseamos no estudo [23], que busca entender como as práticas geralmente associadas a *Continuous Deployment* acharam o seu caminho nas indústrias européia e norte-americana. Nesse estudo os autores utilizaram um método misto de estudo empírico baseado em um pré-estudo na literatura, entrevistas com 20 participantes e um *survey* que recebeu 187 respostas. A ideia era questionar até que ponto o conhecimento na área estava dominado por peculiaridades de um pequeno grupo de grandes empresas, como Facebook e Google.

Os autores também definem a chamada *stairway to heaven* (escada para o céu, em tradução livre), presente na Figura 3.1. Ela tem como objetivo definir um caminho de evolução das empresas para um estágio de entregas sofisticado. A escada permeia práticas de integração contínua, *deployment* contínuo e entregas parciais.

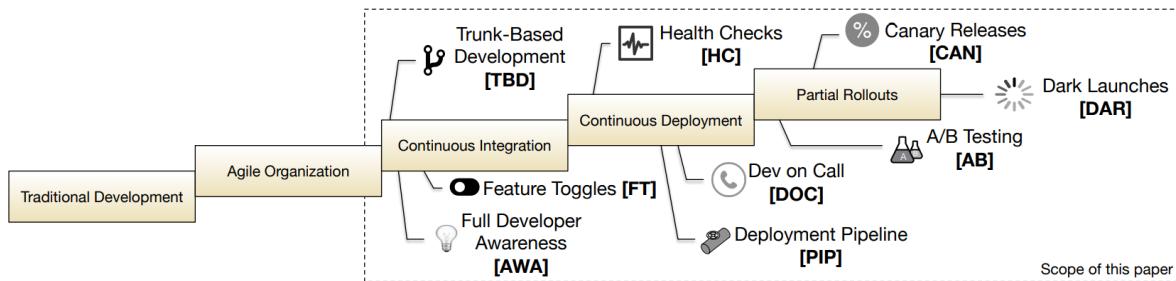


Figura 3.1 A escada de evolução denominada *Stairway to Heaven* proposta pelo artigo base. Fonte: Schermman et al [23]

Através desta metodologia os autores descobriram que, no contexto estudado, problemas arquiteturais são geralmente uma das maiores barreiras para a adoção de CD. Não obstante, a técnica de *Feature Toggles* [17] como forma de realizar entregas parciais adiciona uma complexidade demasiada e não saudável ao código. Por fim, eles concluem que os desenvolvedores necessitam também de um protocolo baseado em princípios que estabeleçam quando deve-se utilizar técnicas de entregas parciais, por exemplo, que funcionalidades e métricas devem ser testadas por um teste A/B [19].

Aprender mais sobre as dificuldades que outras empresas não globais sofrem no processo de adoção de práticas de CI/CD pode gerar mais estudos a respeito de como solucionar tais problemas, assim como mostrar possíveis oportunidades de melhoria e revisão dos processos utilizados. Um trabalho que utilize uma adaptação da metodologia aplicada a uma amostra de um outro contexto pode revelar ainda discrepâncias e semelhanças entre os dois ambientes de

estudo. As discrepâncias levantariam pontos de questionamento, pesquisa e até possíveis melhorias aos ambientes envolvidos. Já as semelhanças podem assegurar que as práticas utilizadas já acharam o seu lugar na indústria e funcionam bem assim como a teoria propunha.

Assim, a grande adoção da indústria mundial das práticas de CI/CD, aliado a falta de trabalhos a respeito no contexto recifense, além da metodologia já validada proposta pelo artigo foram as principais motivações para o desenvolvimento deste trabalho

3.4 Perguntas de pesquisa

Com o intuito de entender como as técnicas de integração, entrega e *deployment* contínuos foram importadas para as empresas recifenses, as seguintes perguntas de pesquisa foram formuladas:

1. Quais as práticas de CI/CD são utilizadas pelas empresas em Recife?
2. O cenário de CI/CD nas empresas em Recife segue o *stairway to heaven* proposto no artigo?
3. Quais são os princípios e práticas subjacentes que governam a adoção de CI/CD na indústria?

Para responder às perguntas acima foi decidido aplicar a pesquisa qualitativa do artigo [23] com desenvolvedores recifenses, baseando-se também na mesma lista de definições de cada uma das práticas envolvidas no *Stairway to Heaven*. A pesquisa qualitativa neste contexto foi considerada fundamental para garantir que os entrevistados entendessem claramente as perguntas e excluir possíveis entendimentos errados de termos em inglês, linguagem não nativa de todos os participantes. Vale salientar que não foi replicado a pesquisa quantitativa presente no artigo base devido à falta de tempo hábil para tal, mas esta pode servir como um trabalho futuro a este.

Além disso, as perguntas 1 e 3 são uma adaptação das perguntas 1 e 2 do artigo base [23], respectivamente, incluindo a técnica de *Continuous Integration*. Não obstante, uma terceira pergunta foi adicionada (*RQ2*), que foca na escada de evolução proposta pelo artigo. Ela tenta responder se a *stairway to heaven* é seguida no contexto de Recife, visto que, no outro, os próprios autores já refutaram esta definição com os seus resultados.

CAPÍTULO 4

Metodologia

Para responder as perguntas introduzidas na seção anterior, foi realizada uma pesquisa qualitativa através de entrevistas semi-estruturadas utilizando como base o roteiro desenvolvido pelo artigo base [23]. A primeira seção abordará uma visão geral sobre os primeiros passos da pesquisa. Já a segunda seção discorre sobre como as entrevistas foram conduzidas. Na terceira, encontra-se todo o processo utilizado para a análise dos dados obtidos na entrevista. Por último, a quarta seção mostra informações a respeito dos participantes da pesquisa.

4.1 Pré-estudo

Com o objetivo de entender melhor sobre como as práticas de CI/CD migraram para as empresas sediadas em Recife, essa fase do estudo contou com a participação de um segundo pesquisador para discussão do artigo base [23]. Na discussão foi definido que seria aplicado apenas o estudo baseado em entrevistas semi-estruturadas para garantir o entendimento dos termos pelos entrevistados e a adequação com as definições do artigo base.

Após a discussão, os pesquisadores acessaram o apêndice do trabalho para obter o guia de entrevista utilizado. Como forma de manter a consistência, foi utilizado o mesmo guia de entrevistas, assim como as mesmas definições utilizadas pelos autores. No início foi montado um arquivo com a definição de cada um dos termos envolvidos em língua portuguesa para ser utilizado como consulta caso haja alguma dúvida de definição de tópicos entre os entrevistados.

Após isso, o guia de entrevistas do estudo base também foi traduzido para português e utilizado durante as entrevistas. Um ponto importante a respeito da tradução é o fato de que alguns termos ainda foram mantidos na língua inglesa devido ao fato de serem conhecidos mundialmente nesta língua. Por exemplo, *Continuous Integration*, *Canary Releases* e *Health check*. O questionário utilizado está presente no [apêndice A](#) deste trabalho em sua versão traduzida.

4.2 Estrutura da Entrevista

O guia de entrevistas se baseia na estratégia de evitar perguntas diretas (exemplo: “determinada prática está sendo utilizada?”). Esse modelo é essencial para garantir que a comparação entre participantes a respeito do uso ou não de práticas está sendo feita de maneira concisa, e não baseada nos conhecimentos prévios de cada participante. Assim, através de perguntas sobre o processo utilizado pelo entrevistado é possível, com uma certa margem de erro associada,

afirmar que práticas ele utiliza. A margem de erro surge do fato de o autor ter que refletir sobre as informações recebidas e as definições para inferir o uso ou não de certa prática.

A entrevista é dividida em 5 sessões:

1. Processo de entrega no geral
2. Papéis/Responsabilidades
3. Garantia da Qualidade (Quality Assurance)
4. Gerenciamentos de Problemas
5. Avaliação de Entrega

No guia, todas as sessões iniciam com uma questão aberta. As entrevistas seguiram os tópicos abordados em cada uma delas, mas sem ordem específica, respeitando o desenrolar da conversa. A primeira entrevista foi guiada pelos dois pesquisadores para assegurar que as próximas seriam feitas de forma semelhante pelos dois. Esta entrevista foi considerada como válida na análise de dados visto que não houve nenhuma mudança no questionário de entrevistas; o próprio já tinha sido validado no artigo utilizado de base para este trabalho. As outras 10 – totalizando 11 entrevistas feitas – foram guiadas por apenas um entrevistador, sendo 5 feitas por cada um dos dois pesquisadores.

Todas as entrevistas ocorreram de forma online, através do Google Meet, e aconteceram entre os meses de Setembro e Outubro de 2020 em português brasileiro. As entrevistas levaram entre 30 e 65 minutos, duração bem parecida com os tempos obtidos no artigo base (35 a 60 minutos), totalizando 7 horas e 15 minutos. Cada uma delas foi gravada pela plataforma para futura análise e 4 delas foram transcritas para o uso em citações neste trabalho, escolhidas através da relevância da entrevista e da forma como certos termos e processos foram apresentados pelo entrevistado. Foi deixado claro em cada uma das conversas a respeito da gravação e que estas seriam utilizadas apenas pelos entrevistadores, respeitando assim o anonimato do participante e confidencialidade de demais informações pessoais, bem como da empresa onde o participante trabalha.

4.3 Participantes

No total foram entrevistados 11 desenvolvedores (P1 a P11) de 7 empresas diferentes sediadas em Recife. A Figura 4.1 mostra todos os dados demográficos obtidos de cada um dos entrevistados. Esses dados foram obtidos pelos entrevistadores no começo de cada entrevista e agregados nesta figura. O tamanho da empresa é definido pela quantidade de colaboradores: *Startup* define empresas com menos de 50 trabalhadores; *SME* são empresas que contém entre 50 e 500 colaboradores; e *CORP* são empresas com mais de 500 colaboradores.

Da amostra, 3 eram mulheres. Pode-se ver a distribuição dos participantes por gênero na Figura 4.2. Dentro desse grupo, 9 trabalhavam com aplicações Web, enquanto 1 trabalhava com sistemas embarcados e o último, com jogos.

ID	Empresa	Tipo de Aplicação	Tamanho da Empresa	Experiência (Anos)	Sexo
P1	E1	Web	SME	2,5	M
P2	E2	Web	CORP	22	M
P3	E2	Web	CORP	2	F
P4	E7	Hardware	CORP	3	F
P5	E5	Games	STARTUP	2	M
P6	E6	Web	STARTUP	1	M
P7	E3	Web	STARTUP	0,58	M
P8	E1	Web	SME	2	M
P9	E3	Web	STARTUP	1	M
P10	E4	Web	SME	1	F
P11	E4	Web	SME	8	M

Figura 4.1 Detalhamento dos entrevistados da amostra, contendo todos os dados demográficos coletados.

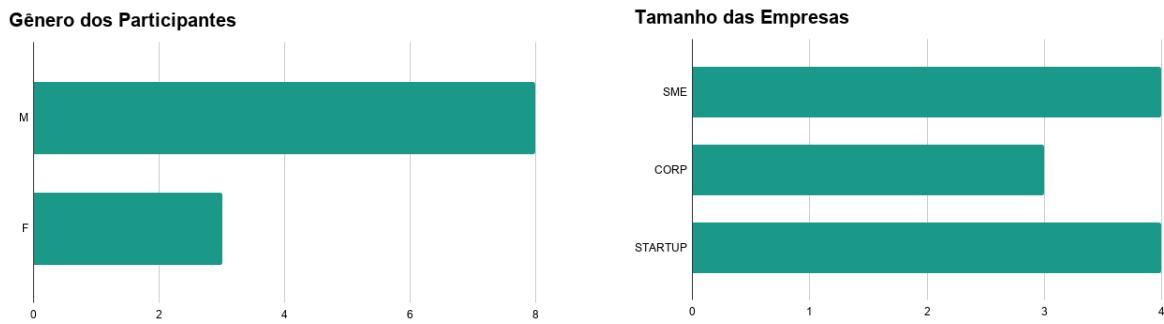


Figura 4.2 Distribuição dos Participantes por gênero

Figura 4.3 Distribuição dos Participantes por tamanho da empresa.

A escolha dos entrevistados foi feita baseado na rede de conhecidos dos entrevistadores, com o propósito de agregar pessoas que trabalhavam em empresas de tamanhos distintos para garantir uma variedade de parâmetros envolvidos. Como é possível perceber na Figura 4.3, com relação a esse aspecto a amostra está bem distribuída.

4.4 Análise dos Dados

A Figura 4.4 apresenta uma visão geral de como funcionou o processo de coleta e análise de dados. A análise foi feita apenas pelo autor deste estudo. O processo escolhido foi baseado nas fases de *Coding* e *Thematic Analysis* da metodologia *Grounded Theory* [14]. No processo de *Coding* a ideia é levantar rótulos ou tags relevantes para o texto e, tradicionalmente, é feito baseado na transcrição das entrevistas. No entanto, neste trabalho o autor gerou códigos através da escuta das entrevistas. Então, como um exemplo, a seguinte citação:

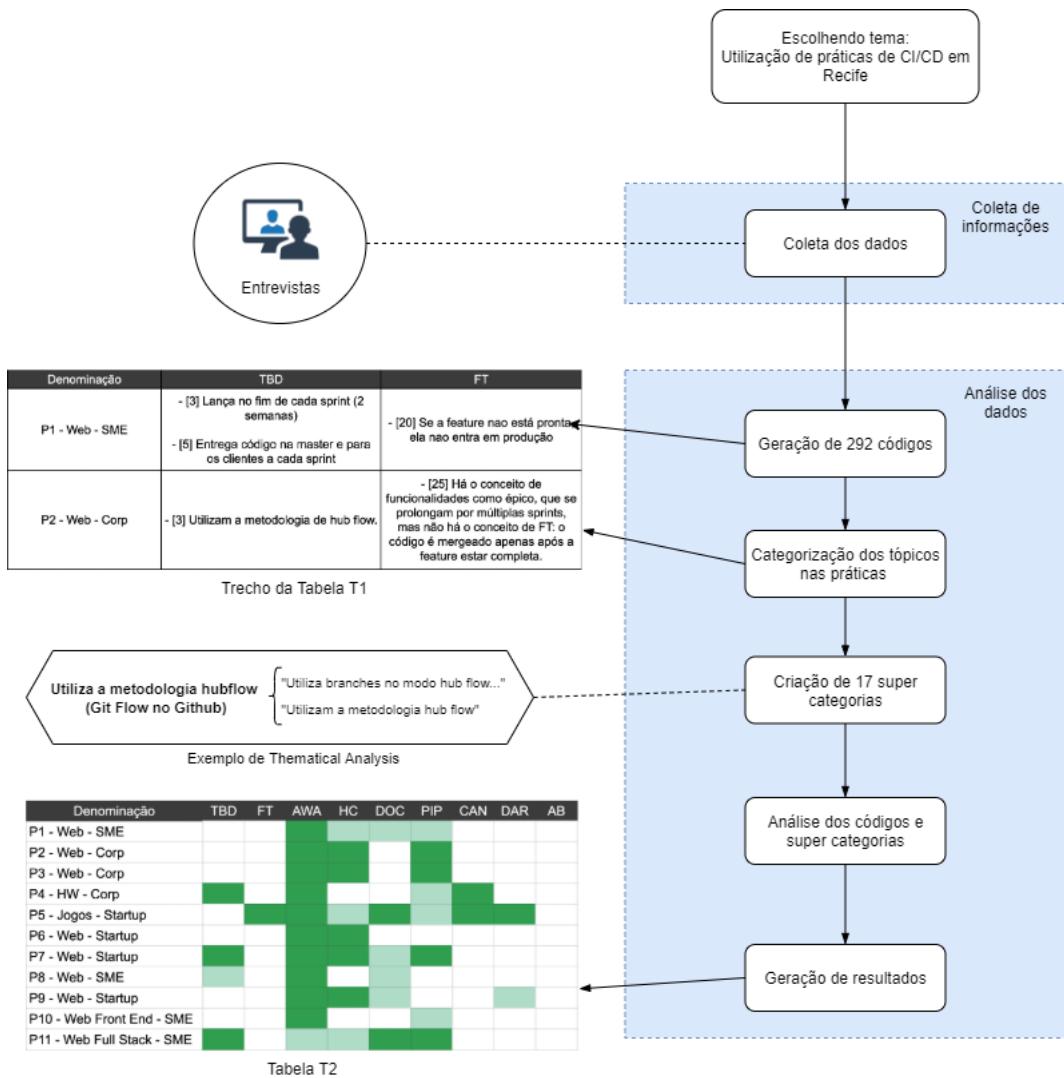


Figura 4.4 Demonstra como funcionou os processos de coleta e análise de dados.

Como somos uma equipe muito pequena, todos os desenvolvedores são meio que DEVOPS. Quando tem que tomar alguma decisão nós entramos em discussão e definimos por nós.

—P5

Gerou o código: “*Todos os desenvolvedores são devops.*” – P5_15, onde P5 identifica o número do participante do qual a entrevista gerou o código, e o 15 define a ordem de criação deste, além de servir como identificador único.

Então, após levantados todos os códigos de uma entrevista, estes foram revisados para garantir semântica e sintaxe adequadas. Alguns códigos nessa fase foram eliminados por redundância, enquanto outros foram quebrados em múltiplos. Depois, eles foram agrupados, quando

compatíveis, em cada uma das 9 práticas descritas pelo artigo base e foi escolhida uma nota entre 0 a 2, representando não utiliza, utiliza parcialmente e utiliza completamente baseado nas definições do artigo base traduzidas. Por exemplo, na Figura 4.4 pela etapa de “Categorização dos tópicos nas práticas”. Este processo foi então replicado para cada uma das 11 entrevistas.

Ao final do processo ainda haviam 3 entrevistas que não geraram nenhum código relacionado a prática de *Dark Launch*. Para estes casos, foi enviado um email diretamente para cada um dos entrevistados com a definição do artigo base da técnica e foi perguntado se o entrevistado utilizava ou não a mesma, o que gerou mais 3 códigos. Ao final, 292 códigos surgiram ao todo.

Após a etapa de geração e revisão dos códigos, estes foram então agrupados, quando válido, na prática específica com a qual aquele código se relacionava. Este agrupamento gerou a Tabela Entrevistado-Pratica-Codigos, presente no [apêndice B](#) deste trabalho, que mostra uma visão geral de cada relação entre prática e entrevistado, contendo a nota dada e os códigos que justificam a nota. A nota é demonstrada pela cor presente em cada célula: branco significa não utiliza, verde claro mostra que o participante utiliza parcialmente, e verde escuro, totalmente. A tabela contém 147 dos 292 códigos gerados e foi utilizada como base para a maioria das figuras geradas neste trabalho. A Figura 4.5 apresenta um trecho da Tabela Entrevistado-Pratica-Codigos.

Denominação	HC	DOC
P1 - Web - SME	- [17] Não há HC rodando atualmente, mas já houve (está quebrado). Há apenas um check sobre o certificado https.	- [9] Geralmente não fica de plantão depois do deploy para algum problema, apenas quando vai haver uma apresentação para clientes.
P2 - Web - Corp	- [14] Utiliza Health check para monitorar se a aplicação está funcionando como esperado e todas suas conexões (banco de dados e cache)	- [21] Não é necessário o uso de DEV on call, eles confiam bastante no processo de garantia de qualidade.

Figura 4.5 Trecho da Tabela Entrevistado-Pratica-Codigos, com os códigos derivados das entrevistas agrupados por prática. As cores de cada célula representam a nota dada para determinada prática e foram dadas com base nos códigos presentes nela. Em algumas células há mais de uma evidência.

Para identificar padrões e abstrações nos códigos agrupados na Tabela Entrevistado-Pratica-Codigos ([apêndice B](#)), foi feito um trabalho de agrupamento semântico, gerando por fim 17 novas super categorias através do processo de *Thematic Analysis* [14]. Esse processo é ilustrado pela etapa de “Criação de super categorias” na Figura 4.4. As super categorias tinham como objetivo agrupar códigos semelhantes ou de mesma semântica para facilitar na reflexão dos dados. Um exemplo de super categoria gerada para a prática de *Trunk Based Development* [TBD] pode ser encontrado na Figura 4.6. A tabela com todas as super categorias geradas pode ser encontrada no [Apêndice C](#).

É importante salientar que ainda durante o processo de levantamento de códigos surgiram

Categories	Codes
Utiliza a metodologia hubflow (Git Flow no Github)	P2 - [3] Utilizam a metodologia de hub flow.
	P5 - [11] Utiliza branches no modo hub flow. Apenas entra na branch master aquilo que está preparado para ir para produção.

Figura 4.6 Exemplo de super categoria gerada durante o processo de *Thematic Analysis*.

tópicos relevantes que não se relacionavam diretamente com as práticas descritas, mas que são perguntados pelo questionário e relevantes para o tema tratado. Surgiram então 2 novos tópicos que agregam códigos sobre as práticas de *Code Review* e de testes automáticos. Para estas foram relacionados 29 códigos dos 292 já mencionados, e duas super categorias foram geradas. A tabela com todos os códigos relacionados está presente no [apêndice B](#) juntamente com as outras práticas.

Por fim, foram geradas algumas tabelas derivadas da Tabela Entrevistado-Pratica-Codigos, ilustrado no passo de “Geração de resultados” na Figura 4.4, para facilitar a visualização e análise dos dados agregados, mostrando um mapa de utilização das práticas na amostra de diferentes formas. Essas tabelas serão apresentados na próxima seção.

CAPÍTULO 5

Resultados

Neste capítulo será apresentado os resultados encontrados com as entrevistas baseado no processo de análise de dados apresentada no capítulo anterior. Primeiramente será abordado como foi feita o agrupamento dos códigos gerados com as práticas. Após isso será focado o estudo do predomínio das práticas em Recife, seguido da análise de coerência do stairway to heaven. Por fim, serão apresentados princípios e práticas subjacentes que governam a adoção de CI/CD na amostra.

5.1 Estudo de Predomínio das Práticas

Com o intuito de responder a pergunta de pesquisa RQ1 – *Quais as práticas de CI/CD são utilizadas pelas empresas em Recife?* – de forma restrita aos 11 entrevistados das 7 empresas da amostra, foi montada a Tabela 5.1, demonstrando a utilização de cada uma das práticas definidas para cada um dos participantes. Nesta, é demonstrada através da coloração da célula o grau de utilização de determinada prática por determinado participante. Assim, o verde mais escuro significa que o participante utiliza totalmente, enquanto o verde claro significa utilização parcial e, por fim, o branco denota a não utilização. Esta foi produzida a partir da Tabela Entrevistado-Pratica-Codigos ([apêndice B](#)), retirando-se os códigos agrupados e ordenando as colunas pela utilização de determinada prática. A ideia é replicar a Tabela 1 do artigo base [23], presente na Figura 5.1.

Com a Tabela 5.1 é possível perceber que *Developer awareness [AWA]* foi a prática mais encontrada em toda a amostra, sendo totalmente utilizada pela grande maioria dos entrevistados; apenas um deles utiliza parcialmente. Logo após, pode-se encontrar as práticas de *Health Check [HC]* e *Deployment Pipeline [PIP]* na segunda e na terceira colocação, respectivamente. No geral, também pode-se inferir que as técnicas de *Partial Rollouts* são ainda muito pouco utilizadas, com as 3 práticas do grupo entre as quatro últimas colocadas. Vale a pena citar que nenhum dos entrevistados utiliza, mesmo que precariamente, a técnica de Testes A/B [AB].

Quando comparamos a Tabela 5.1 com a Tabela 1 do artigo base (Figura 5.1), podemos perceber que há diferenças de posição entre práticas, mas que não há mudanças exorbitantes. Com a ajuda da Tabela 5.2, é possível perceber que há uma diferença de no máximo 2 posições na ordem de predomínio, ao comparar os resultados obtidos neste trabalho com os do artigo base. É possível perceber que as práticas de AWA, TBD, DAR e FT tiveram mudanças de 2 posições, enquanto HC, PIP, CAN e AB tiveram apenas diferença de 1 unidade de posição. Apenas *developer on call* permaneceu no mesmo local dentro das duas amostras. Por fim, é interessante perceber que o conjunto das três primeiras práticas é o mesmo, mas em posições

Denominação	AWA	HC	PIP	DOC	TBD	CAN	DAR	FT	AB
P1 - Web - SME									
P2 - Web - Corp									
P3 - Web - Corp									
P4 - HW - Corp									
P5 - Jogos - Startup									
P6 - Web - Startup									
P7 - Web - Startup									
P8 - Web - SME									
P9 - Web - Startup									
P10 - Web Front End - SME									
P11 - Web Full Stack - SME									

Tabela 5.1 Nível de utilização de cada uma das práticas, com as colunas ordenadas em ordem decrescente de uso. Práticas: AWA: *Developer Awareness*; HC: *Health Check*; PIP: *Deployment Pipeline*; DOC: *Developer on Call*; TBD: *Trunk Based Development*; CAN: *Canary Releases*; DAR: *Dark Launches*; FT: *Feature Toggles*; AB: Testes A/B.

trocadas no dois estudos.

5.2 Stairway to heaven

Com o objetivo de responder a pergunta de pesquisa RQ2 – *O cenário de CI/CD nas empresas em Recife segue o “stairway to heaven” proposto no artigo?* – foi produzida a Tabela 5.3. Esta contém a visualização de utilização das práticas, utilizando a mesma técnica de cores aplicada na Tabela 5.1 e explicada na seção anterior, mas com as colunas ordenadas pela escada definida pela sequência de práticas do *Stairway to Heaven* apresentada na Figura 3.1 (Capítulo 3).

Com a Tabela 5.3 é possível perceber que a amostra deste estudo, assim como a amostra do estudo original, não segue a evolução proposta pelos autores do artigo base. Isso fica claro quando percebe-se que não há, em nenhum dos entrevistados, uma relação clara entre a coloração da coluna com a sua anterior. É possível notar também que há uma lacuna nas duas primeiras práticas, seguido de uma grande utilização da terceira, confirmando a tese de que a *Stairway to Heaven* não foi identificada na amostra. O mais próximo dos participantes é P5, mas ainda há nele a falta do pilar TBD do primeiro degrau escada. É interessante destacar que na amostra do próprio artigo base também não foi possível identificar a escada de evolução, como é possível visualizar na Figura 5.1.

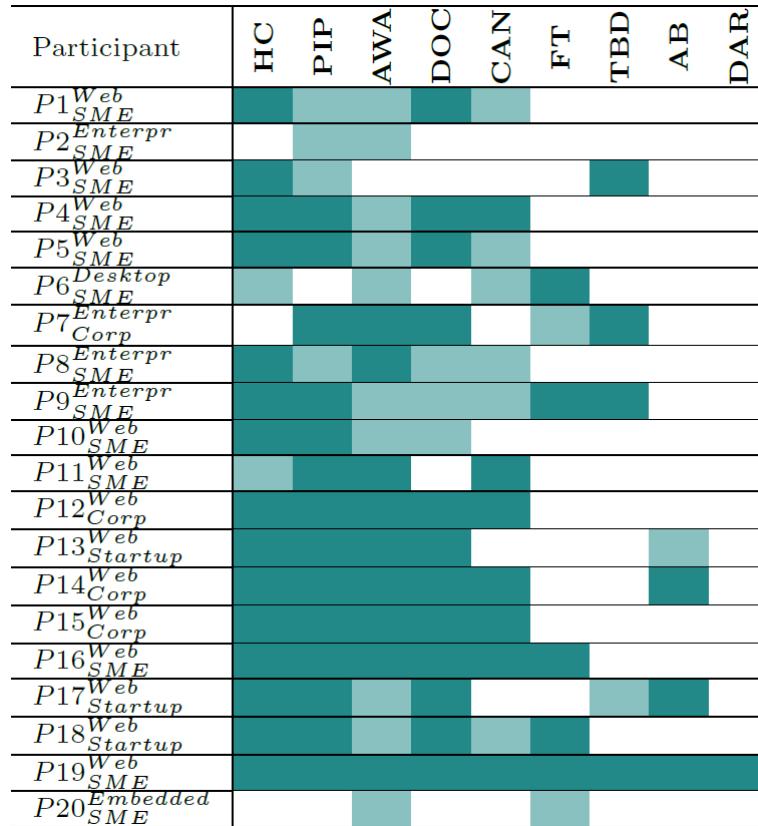


Figura 5.1 Utilização das práticas pelos participantes do artigo base. Fonte: Schermann et al [23]. Práticas: AWA: *Developer Awareness*; HC: *Health Check*; PIP: *Deployment Pipeline*; DOC: *Developer on Call*; TBD: *Trunk Based Development*; CAN: *Canary Releases*; DAR: *Dark Launches*; FT: *Feature Toggles*; AB: Testes A/B.

5.3 Análise das Práticas

Nesta seção será abordado, para cada uma das práticas, as principais curiosidades encontradas na amostra. Esta tem como objetivo responder a pergunta RQ3: *Quais são os princípios e práticas subjacentes que governam a adoção de CI/CD na indústria?* As seguintes subseções agrupam as práticas emergentes através das práticas gerais definidas no *Stairway to Heaven*, inclusive seguindo a ordem deste, conforme explicado no Capítulo 4, Seção 4.4.

Para poder definir padrões encontrados na amostra o autor, após a fase de agrupamento semântico, juntou todos os códigos e super categorias ligadas a cada uma das práticas baseando-se principalmente na nota de utilização. Com a leitura e reflexão deste conjunto, o autor procurou por princípios que governam a adoção ou não de cada prática. Foi feito também um estudo comparativo com os resultados obtidos pelo artigo base para analisar discrepâncias e congruências entre os dois contextos de estudo.

Prática/Trabalho	Contexto Recifense	Artigo Base	Diferença de Posição
AWA	1º	3º	2
HC	2º	1º	1
PIP	3º	2º	1
DOC	4º	4º	0
TBD	5º	7º	2
CAN	6º	5º	1
DAR	7º	9º	2
FT	8º	6º	2
AB	9º	8º	1

Tabela 5.2 Diferença entre o artigo base e este trabalho a respeito da ordem de predomínio das práticas. Práticas: AWA: *Developer Awareness*; HC: *Health Check*; PIP: *Deployment Pipeline*; DOC: *Developer on Call*; TBD: *Trunk Based Development*; CAN: *Canary Releases*; DAR: *Dark Launches*; FT: *Feature Toggles*; AB: Testes A/B.

5.3.1 Integração Contínua

Com a Tabela 5.1 é possível perceber que, apesar da grande disseminação a respeito de informações relacionadas a infraestrutura e manutenção de software – ligadas à prática de *Developer Awareness* [22] – estar em primeiro lugar, as outras técnicas que compõem o conjunto de *Continuous Integration* ainda não estão disseminadas na nossa amostra.

Nesta subseção, serão abordadas a seguir cada uma das práticas ligadas ao processo de Integração contínua, como explicado no Capítulo 2: *Trunk Based Development* [TBD] [9], *Feature Toggles* [FT] [17] e *Developer Awareness* [AWA].

5.3.1.1 Trunk Based Development [TBD]

Na amostra é possível perceber que a técnica de *Trunk Based Development* [TBD] não é tão amplamente adotada, visto que apenas 4 entrevistados utilizam ao menos parcialmente. Destes, 2 comentam que entregam novas versões aos clientes baseados em novas funcionalidades, e não em *sprints*. Já entre os que não utilizam, 5 integram o código apenas no final da *sprint*. Deste grupo, 2 utilizam a metodologia *Git Flow* [8], que define uma maneira de manusear várias branches ao mesmo tempo de modo que os desenvolvedores deparam com o mínimo de conflitos possível e que software seja entregue em versões bem definidas.

5.3.1.2 Feature Toggles [FT]

No estudo foi possível perceber que, na amostra, a prática de *Feature Toggles* [FT] é raramente utilizada, assim como no artigo base. Esta técnica foi encontrada apenas na equipe do participante P5, que a utilizava para esconder funcionalidades enquanto testes manuais ainda estavam sendo feitos. Geralmente esta técnica é utilizada para auxiliar a integração de códigos ainda não finalizados quando a equipe utiliza técnicas como o *trunk based development*, mas este

Denominação	TBD	FT	AWA	HC	DOC	PIP	CAN	DAR	AB
P1 - Web - SME									
P2 - Web - Corp									
P3 - Web - Corp									
P4 - HW - Corp	■								
P5 - Jogos - Startup		■							
P6 - Web - Startup									
P7 - Web - Startup	■								
P8 - Web - SME		■							
P9 - Web - Startup								■	
P10 - Web Front End - SME									
P11 - Web Full Stack - SME	■								

Tabela 5.3 Nível de utilização de cada uma das práticas, com as colunas ordenadas na ordem do *Stairway to Heaven*. Práticas: AWA: *Developer Awareness*; HC: *Health Check*; PIP: *Deployment Pipeline*; DOC: *Developer on Call*; TBD: *Trunk Based Development*; CAN: *Canary Releases*; DAR: *Dark Launches*; FT: *Feature Toggles*; AB: Testes A/B.

não é o caso de P5. É interessante notar que este participante também foi um dos poucos que utilizava a técnica de *Canary Releases* [CAN].

Entre o grupo dos que não utilizavam, 7 só enviam código novo para produção quando a funcionalidade está concluída. Deste grupo, 2 comentaram que fazem uso de conceito de épicos, onde uma história de usuário se prolonga por mais do que apenas uma sprint. É também interessante notar que P3 está em vias de utilizar esta técnica para reduzir conflitos de *merge* e *rebase* devido ao grande número de desenvolvedores em seu time, como podemos ver na citação:

“O meu time tem 23 [pessoas]. [...] a gente tá trabalhando em funcionalidades muito distintas, então às vezes acontece de termos um paralelismo de branches muito grande. [...] é muito complicado ‘mergear’ e fazer rebase de tudo. Realmente dá muitos conflito”

—P3 – WEB – CORP

5.3.1.3 Developer Awareness [AWA]

Na amostra é possível identificar que a prática de *Developer Awareness* [AWA] é amplamente adotada nas companhias, assim como na amostra do artigo base. Em 6 entrevistas foi possível perceber que o time de desenvolvimento era o mesmo responsável pela entrega e manutenção da aplicação. Em outras 2, há um time específico de *DevOps*, mas não havia grandes silos entre este e a equipe de desenvolvimento. Um caso interessante foi o de P11, que, apesar de

um conhecimento espalhado dentro da equipe, há receio e insegurança por parte de alguns a respeito de questões de infraestrutura no geral.

5.3.2 Deployment Contínuo

Nesta subseção, serão abordadas as práticas ligadas ao processo de *Deployment* contínuo, como explicado no Capítulo 2: *Health Checks* [HC] [20], *Developer on Call* [DOC] [9] e *Deployment Pipeline* [PIP] [20]. É possível inferir, baseado principalmente na Tabela 5.1, que as técnicas ligadas ao processo estão presentes na maioria das equipes da amostra: as 3 estão nas quatro primeiras colocações.

5.3.2.1 Health Checks [HC]

Sobre a prática de *Health Checks* [HC], é possível perceber que, apesar de não ser o mais adotado – como foi no artigo base, ainda tem destaque entre as outras, presente na segunda colocação. Na amostra, 7 entrevistados continham pelo menos uma forma rudimentar de verificação e alertas, e destes, 2 continham apenas verificações não muito complexas. Um ponto interessante que surgiu foi o fato de P4 achar que não era necessário utilizar esta técnica por estar em fase de prototipação.

5.3.2.2 Developer on Call [DOC]

Na amostra é possível perceber que a técnica de *Developer on Call* [DOC] é mais adotada de forma implícita do que de fato definida – 3 entrevistados estão em times que funcionam desta forma. Contudo, 5 pessoas do grupo não utilizam esta prática: alguns comentaram que a confiança nos testes automáticos faz com que não utilizem a prática, enquanto outro comentou que a prática é inclusive mal vista pela empresa.

Uma dicotomia interessante foi encontrada entre os resultados da amostra deste trabalho e o do artigo base. Este último comenta que a prática já está sendo largamente aceita nas organizações atualmente, e inclusive um dos entrevistados comenta que essa responsabilidade de ficar até mais tarde para resolver problemas leva os desenvolvedores a escrever e testar seus códigos mais veemente. Tal argumentação tem como base a seguinte citação retirada do artigo e traduzida:

“Se você não tem testes suficientes e faz deploy de um código ruim isso vai se voltar contra você pois você estará de plantão e terá que dar suporte a isto.”

—P14 (DO ARTIGO BASE) – WEB – CORP

Contudo, com a seguinte citação de P2, é possível perceber que há um sentimento contrário: confia-se no processo de qualidade e, por isso, não necessitam de plantão.

“Temos o ciclo de QA, se encontrar alguma coisa a gente vê [...] não precisa isso de plantão não.”

—P2 – WEB – CORP

5.3.2.3 Deployment Pipeline [PIP]

Em relação à prática de *Deployment Pipeline* [PIP] é possível inferir que ela é amplamente adotada pela amostra, visto que apenas 2 entrevistados obtiveram nota 0 (não utiliza). No artigo base é possível perceber um resultado semelhante a este. Uma grande parte dos entrevistados segue o mesmo padrão para qualquer tamanho da mudança. Outros 2 tinham alguns processos automatizados, mas a *pipeline* era diferente dependendo do tamanho da mudança.

É interessante perceber que alguns times ainda demonstram falta de automação dos processos da *pipeline*. Isto acontece em decorrência da complexidade da automação, devido às tecnologias e ferramentas utilizadas, ou da falta de prioridade do time para tal.

5.3.3 Entregas Parciais

Nesta subseção, serão abordadas as práticas ligadas ao processo de Entregas Parciais, como explicado no Capítulo 2: *Canary Releases* [CAN] [18], *Dark Launches* [DAR] [9] e Testes A/B [AB] [19]. Na amostra foi possível perceber que todas as técnicas são muito pouco utilizadas: todas estão entre as 4 últimas colocações. É valido também notar que as 3 mantiveram a mesma ordem de utilização do processo do artigo base [23].

5.3.3.1 Canary Releases [CAN]

A técnica de *Canary Releases* [CAN] apareceu nos contextos de jogos e no de sistemas embarcados. Na equipe do entrevistado P5, que trabalha no domínio de jogos eletrônicos, os principais jogadores – conhecidos como “baleias” – são escolhidos para participar de um *early access* de novas funcionalidades. Esses testes levam em torno de 1 semana.

Já na equipe de P4, que trabalha com sistemas embarcados, a prática era necessária devido ao contexto de atuação e às tecnologias utilizadas. Como o sistema que está em fase de prototipação servirá para o contexto médico, vários testes de campo deveriam ser feitos para garantir que todas as funcionalidades estivessem de acordo com o esperado. Para os testes, o cliente que contratou a empresa de P4 escolhia a quantidade de pessoas e local que serviria como validação de funcionalidades.

Entre os entrevistados que não utilizam a prática de *Canary Releases*, 3 têm um ambiente de homologação para testes e validação de requisitos, mas este utiliza dados diferentes dos de produção. Outros 3 comentam que as features são sempre entregues para todos os usuários ao mesmo tempo.

5.3.3.2 Dark Launches [DAR]

Do grupo das práticas associadas a Entregas Parciais, *Dark Launches* [DAR] foi a segunda mais utilizada, mas a menos conhecida entre os entrevistados. Isto se confirma com o fato de

que 6 entrevistados disseram nunca ter utilizado, e 3 destes disseram especificamente que não conheciam a técnica. Importante notar que no estudo base [23] esta é a menos utilizada do grupo de práticas.

Dark Launches foi utilizado totalmente por P5 no contexto de jogos para testes manuais, e apenas parcialmente no contexto de WEB por P9 para validações de alguns cenários que dependiam de dados de produção.

“A gente implementa a funcionalidade mas condiciona a não aparecer para o usuário até que a gente queira.”

—P5 – JOGOS – STARTUP

5.3.3.3 Testes A/B [AB]

A prática de Testes A/B [AB] não foi identificada em nenhum dos participantes. Entre as principais causas levantadas para tal, 4 entrevistados comentaram que não utilizam pela baixa quantidade de usuários ativos no sistema. Outros 2 comentaram que a técnica não se aplicava ao contexto da aplicação. É interessante notar que esses dois motivos estão presentes no artigo base [23], contudo, ao contrário deste, ninguém na amostra comentou sobre problemas na arquitetura como causa para não utilização.

“O sistema da gente – apesar de lidar com uma massa de dados muito grande – não têm tantos usuários, então não faz muito sentido [utilizar testes A/B]....”

—P2 – WEB – CORP

Outro motivo importante foi levantado por P8, que comenta que aparentemente não existe na empresa o interesse em investir nesta prática. P3 comenta ainda que o time está mais focado em entregar novas funcionalidades, pois a demanda por parte do cliente está muito grande.

5.4 Descobertas adicionais

Esta subseção abordará sobre os dois tópicos marginais que foram levantados durante a análise dos dados obtidos pelas entrevistas: *Code Review* e testes automáticos.

5.4.1 Code Review

A prática de *Code Review* [5] é bem vista pela maioria dos entrevistados, considerado uma técnica importante e essencial para o controle de qualidade de código. As razões para o uso são diversas, desde de impor boas práticas – por P7 – até o compartilhamento e nivelamento do conhecimento – por P11.

“... a partir do momento que o sistema vai crescendo, o próprio desenvolvedor não tem a noção de que aquela sua mudança não é a melhor forma de fazer e que pode quebrar outras partes do sistema...”

—P2 – WEB – CORP

Ainda na amostra, 3 entrevistados acreditam que a prática funciona principalmente para troca de conhecimento. Sobre isso, a entrevistada P10 comenta que agrupa muito valor para ela como novata no time, mas acha que adiciona um tempo desnecessário na entrega de funcionalidades por pessoas mais experiente, visto que – para ela – a técnica não faria sentido neste caso.

Outro ponto importante foi a distinção entre dois relatos a respeito do engajamento do time com a prática. Enquanto P3 comenta que a equipe dela é bem aberta ao debate e está engajada com o processo, P7 fala que o processo funciona “mais ou menos”, dependendo do humor dos revisores.

5.4.2 Testes Automáticos

Testes automáticos são, no geral, bem vistos e extremamente recomendados pela maioria dos entrevistados como forma de prevenir erros em tempo de execução. Contudo, o participante P2 comenta que ainda é contrário a depender somente deles como garantia de qualidade.

“... automação [de testes] não resolve todos os problemas [relacionados a garantia de qualidade], ele vai identificar muita coisa, mas tem várias outras que precisamos do olhar de um testador...”

—P2 – WEB – CORP

Na amostra, 5 entrevistados acreditam que o aumento da cobertura de testes automáticos pode diminuir a quantidade de bugs em produção. Dentro desse grupo, o participante P9 comenta que isto poderia, no entanto, diminuir a velocidade de entregas do time. P5 adicionou ainda que sente que a *sprint* é muito curta e não consegue tempo dentro destas para adicionar testes automáticos.

5.5 Ameaças a validade

Mesmo com a metodologia definida e seguida utilizando métodos conhecidos pela comunidade científica, é necessário levantar possíveis ameaças à validade dos resultados encontrados neste trabalho. Uma ameaça plausível é a quantidade relativamente pequena de entrevistas feitas.

É importante salientar também que os códigos gerados durante o processo de codificação das entrevistas sofreram um certo enviesamento visto que este trabalho é uma replicação de um estudo, então o autor tinha em mente que assuntos estavam sendo procurados na fala durante o levantamento de códigos.

Outra ameaça que deve ser levada em conta é o processo de *coding* realizado. Ele foi feito baseando-se no áudio das entrevistas, e não nos texto transcritos, como geralmente é feito [14]. Isso pode tornar os códigos enviesados ou até mesmo significar a falta de códigos importantes que poderiam ter sido levantados pelo processo original.

CAPÍTULO 6

Conclusão

Com os resultados encontrados, é possível perceber que, apesar de algumas poucas diferenças a amostra deste trabalho se comportou de forma condizente àquela encontrada no artigo base [23]. Desde o ranking de utilização de cada prática até as barreiras enfrentadas pelos desenvolvedores, houve uma congruência razoável entre os dois resultados.

Sobre o processo de integração contínua, pode-se inferir que as técnicas ligadas ao *merge* contínuo estão pouco presentes, apesar de *Developer Awareness* ser a prática mais utilizada da amostra. A maioria entrevistada integra o código na *branch* principal apenas no final da *sprint*.

É possível perceber ainda que todas as práticas ligadas à *Deployment* contínuo estão muito presentes na amostra. Mesmo com a prática de *Developer on Call* sendo mais utilizada de forma implícita do que de fato definida, e com algumas empresas interpretando-a como uma má prática, ela está em 4º lugar no ranking de utilização montado e é a de menor colocação do grupo.

Por fim, foi possível perceber que a amostra utiliza muito pouco as práticas de entregas parciais. Foi possível identificar técnicas desconhecidas por um grupo razoável de entrevistados – *Dark Launches* – e até técnicas que não foram utilizados por nenhum dos participantes – Testes A/B.

6.1 Trabalhos Futuros

Como trabalhos futuros, pode ser feito o mesmo estilo de entrevistas com um grupo maior de pessoas. A reaplicação poderia levantar novas barreiras e até identificar lacunas que não puderam ser vistas com a amostra deste trabalho. Outro ponto que também seria de grande valia é a pesquisa sobre o mesmo tema, mas de forma quantitativa, com o objetivo de entender melhor que práticas estão sendo utilizadas na indústria de Recife com um grupo maior de entrevistados.

APÊNDICE A

Questionário Traduzido

A.1 Demografia

- Qual seu cargo no trabalho atualmente?
- Qual o tamanho da empresa para o qual você trabalha?
- Qual é o tamanho típico das equipes dentro da empresa que você trabalha?
- Quantos anos de experiência profissional em <cargo do entrevistado> você tem?
- Qual o domínio (contexto: educação, saúde, etc) da empresa ou projeto no qual você trabalha?

A.2 Processo de entrega em geral

Dado uma funcionalidade recém-implementada, como é o processo de entrega da sua empresa a partir do commit da funcionalidade até chegar ao ambiente de produção e, portanto, aos clientes?

Possíveis questões complementares:

- O processo é o mesmo para toda mudança no código, isto é, é o mesmo para uma funcionalidade totalmente nova bem como para uma mudança pequena no código? Quem define/decide sobre esse “impacto” das mudanças?
- As mudanças são lançadas diretamente para todos os usuários?
- O quanto automatizado é esse processo?
- Quanto tempo leva desde o commit de uma funcionalidade até que ela esteja disponível para os usuários?
- Qual a frequência típica de entrega da sua empresa? A cada commit, uma vez por dia, uma vez por semana, etc? Como isso se relaciona com os processos de desenvolvimento de software? Por exemplo, você está fazendo entregas “sempre” ou só no fim de uma sprint?

- Como os empregados (incluindo desenvolvedores) ficam informados acerca das entregas, por exemplo, qual versão está atualmente implementada, quais versões/funcionalidade estão sob teste em certos ambientes? Como essa comunicação é gerenciada entre equipes e nas equipes?
- Como você julga pessoalmente seu processo de entrega? O que funciona bem, e o que não funciona?

A.3 Papéis/Responsabilidades

Quais são, tipicamente, os papéis envolvidos no processo de entrega da sua empresa e quem é responsável pelo que?

É uma abordagem mais colaborativa com times contendo pessoal de operações e pessoal de desenvolvimento, ou existem equipes dedicadas, por exemplo: o time de operações assume a partir do momento que as mudanças feitas pelo time de desenvolvimento passaram os testes e estão prontas para serem entregues?

Possíveis questões complementares:

- Quem é responsável por monitorar a entrega uma vez que ela chegou em produção?
- Como você julga, pessoalmente, as definições de papéis na sua equipe? Elas fazem sentido? Algo está faltando, ou não está muito claro?
- Caso haja papel de DevOps: Como DevOps é realizado dentro da sua empresa?

A.4 Garantia da Qualidade

Quando é feito o commit de uma mudança no sistema de controle de versão (git, por exemplo) da sua empresa ou do seu projeto, como vocês garantem que commits errôneos ou commits que não seguem certas orientações/padrões não cheguem em produção? Como vocês garantem a qualidade do software na empresa?

Possíveis questões complementares:

- Caso seja um processo em estágios, quais estágios existem?
- Caso seja em estágios: Testes críticos são selecionados e executados em estágios iniciais para obter feedback mais rapidamente?
- É feito o build do software exatamente uma vez durante o processo todo, ou cada estágio requer uma build com configuração diferente? Tais arquivos de configuração são gerenciados pelos sistemas de controle de versão?
- Existem estágios/barreiras de qualidade que exigem uma aprovação manual explícita?
- Existe um ambiente parecido com o de produção ou alguma forma para que vocês tenham a certeza de que as mudanças vão funcionar em produção também?

- As revisões de código (code reviews) são parte do seu processo de análise de qualidade(testes)? Quais as razões para que isso seja feito/não seja feito?
- Você acha que o processo de análise de qualidade de vocês funciona bem? O que poderia ser melhorado, na sua opinião?

A.5 Gerenciamento de Problemas

Suponha que uma nova funcionalidade ou mudança chegou em produção, mas não se comporta como esperado(*). Quais são os passos executados e por quem são executados para gerenciar problemas em caso de:

1. Bugs devidos a erros no código
2. Defeitos devido a desvios dos requisitos, resultando em, por exemplo, performance ruim, manutenibilidade ruim, usabilidade ruim, etc.
3. Problemas com a disponibilidade de certas funcionalidades/serviços/partes da sua aplicação.

Adicionalmente: Como vocês identificam ou mensuram se o sistema está exibindo um comportamento inesperado?

Possíveis questões complementares:

- Tais problemas são comuns?
- Como são os problemas típicos que ocorrem em tempo de execução?
- O lançamento de uma funcionalidade requer que os desenvolvedores envolvidos estejam em plantão para lidar com tais problemas?
- Como tais problemas são identificados/descobertos? Existe suporte para isso fazendo uso de alguma ferramenta?
- Reversões (rollbacks) são utilizados? Caso afirmativo, quanto a compatibilidade de versões, como vocês lidam com problemas quando revertendo para versões incompatíveis?
- Você tem quaisquer sugestões para como prevenir uma boa parcela de problemas em tempo de execução?
- Em caso de uma arquitetura baseada em serviço: O quanto comum é que novas versões de certos serviços sejam incompatíveis com outros serviços já existentes?

A.6 Avaliação da entrega

Dado a entrega contínua de novas versões, como vocês comparam se a versão mais recente é “melhor que” ou traz benefícios comparada com suas predecessoras? Como vocês avaliam se uma certa feature/mudança:

- Está satisfazendo as demandas dos usuários
- Tem o impacto desejado no lucro, performance, mantinabilidade, usabilidade, ... ?

Suponha que você quer comparar duas versões da sua aplicação que foram entregues e estão em execução. Quais as métricas que você consideraria?

Possíveis questões complementares caso live testing (Canary, A/B, ...) for usado:

- Qual a duração de tais testes?
- Qual a quantidade? Quantos testes rodam em paralelo?
- Cada um desses experimentos testa exatamente uma feature?
- Como esses testes são avaliados, quando, e em quais intervalos?
- Quem define as métricas e as limiares que se deve observar?
- Como você se certifica que os experimentos paralelos não estão influenciando um ao outro?
- Qual o escopo dos testes? Quais usuários são selecionados para tais testes e como eles são selecionados?

A.7 Finalização

Caso não tenha sido perguntado ou mencionado durante a entrevista:

- Que tipo de arquitetura tem o sistema? Monolítica, baseada em serviços (microserviços)?
- Caso teste A/B não seja utilizado: Quais são as razões para não utilizar técnicas como teste A/B?
- Como você lida com funcionalidades que ainda não estão prontas para serem entregues, especialmente se elas necessitam de mudanças mais complexas através da base de código? Você faz uso de condicionais no código que previne tais funcionalidades de serem executadas até estarem prontas?
- Você tem processos de entrega diferentes para projetos diferentes?
- Na sua opinião, o quanto automatizado os processos de entrega devem ser? Por exemplo, deve haver uma aprovação manual antes de fazer uma reversão automática para uma versão anterior (rollback) quando certos limiares não são atingidos?

APÊNDICE B

Tabela Entrevistado-Pratica-Codigos

APÊNDICE B TABELA ENTREVISTADO-PRÁTICA-CÓDIGOS

Demografia	TBD	FT	AMA	HC	DOC	PP	CAN	DAR	AB
P1 - Web - SME	<ul style="list-style-type: none"> - [3] Lança no fim de cada sprint (2 semanas) - [5] Enrega código na master e para os clientes a cada sprint 	<ul style="list-style-type: none"> - [20] Se a feature não está pronta, ela não entra em produção. 	<ul style="list-style-type: none"> - [8] É ele quem faz a entrega nos ambientes. - [10] Há pessoas ele no equipo, e ele é responsável pelo deploy e manutenção dos ambientes. 	<ul style="list-style-type: none"> - [17] Não há HC rodando querido. Ele apenas um check sobre o certificado https. 	<ul style="list-style-type: none"> - [9] Geralmente não fica de planejado depois do deploy para algum problema, apenas quando vai haver uma apresentação para clientes. 	<ul style="list-style-type: none"> - [11] O ciclo o consiste em: Functional testing, reviews e manual em staging, deploy. - [6] Consegue fazer a entrega de forma rápida (máximo de 30 minutos), automaticamente. 	<ul style="list-style-type: none"> - [2] A funcionalidade é lançada para todos que forem afetados, já que o código é baseado na lista de produtóis que foram criados. - [4] Em um horário entre na mesma hora para todos. 	<ul style="list-style-type: none"> - [22] Conhece a técnica de dark launches, mas nunca fez uso dela. 	<ul style="list-style-type: none"> - [7] A funcionalidade é lançada para todos que forem afetados, já que o código é baseado na lista de produtóis que foram criados.
P2 - Web - Corp	<ul style="list-style-type: none"> - [3] Utiliza a metodologia de hub flow. 	<ul style="list-style-type: none"> - [25] Há o conceito de funcionalidades como sprint, que se prolongam por muitas sprints, mas não há o conceito de PR, o código é mantido apenas para a feature mais recente estar completa. 	<ul style="list-style-type: none"> - [8] Existe uma ferramenta de CI/CD que automatiza a criação de algumas coisas facilita o provisionamento de ambientes. - [14] Utiliza Health check para monitorar se a aplicação está funcionando corretamente e todos os serviços que a apoiam estão funcionando como esperado e todas suas conexões (banco de dados e cache). 	<ul style="list-style-type: none"> - [17] Não há developer on Call, mas não é necessário o uso de DEV on call, eles contam bastante no processo de garantia de qualidade. - [7] Ele aceita que podem ter um processo mais automatizado mas não é automatizado. 	<ul style="list-style-type: none"> - [13] Existem ambientes de staging paralelo com produção, mas este é utilizado apenas para testes da equipe interna e apresentações. 	<ul style="list-style-type: none"> - [2] Processo: Desenvolvimento e testes automáticos + Ambiente de staging para check final + PR (Request com code-review). São feitas automática e manualmente a partir de um dev. - [4] Existe um ambiente de staging similar merge + teste no PR. - [6] Bugs seguem o mesmo processo de features. 	<ul style="list-style-type: none"> - [5] Dá para gerar a release, ela é homologada para o cliente. Ao fim da sprint, essa release é levada para produção. 	<ul style="list-style-type: none"> - [24] Não utiliza AB pois a infra hoje tem apenas um servidor único onde todos os usuários acessam o site da sua vez. - [26] Não conhece a técnica de dark launches. 	<ul style="list-style-type: none"> - [4] Em um horário entre na mesma hora para todos.
P3 - Web - Corp	<ul style="list-style-type: none"> - [4] O código de cada sprint é colocado na branch principal assim no final da mesma. 	<ul style="list-style-type: none"> - [6] Tem a ideia de fazer uma funcionalidade entre duas sprints, mas nunca entregam pela metade, é sempre apenas quando ela estiver completa. - [8] Há uma certa interação e estudo para uso de feature flags, mas ainda não foi implementado, está na fase de estudo. - [11] O deploy é feito pelo projeto, não por JIRA. 	<ul style="list-style-type: none"> - [7] Durante há uma nova versão no ar e envendo um email para todos os interessados. - [9] Há uma equipe dedicada para devops. Mas sempre temos uma conversa entre a equipe do projeto e o SRE para melhorar o produto final. 	<ul style="list-style-type: none"> - [29] Utiliza Health Checks para garantir que o sistema de produção está disponível e que a conexão com o banco de dados está funcionando. 	<ul style="list-style-type: none"> - [23] Não há developer on Call. Geralmente o que acontece é que a pessoa que fixa a funcionalidade resolve o bug, mas não tem ninguém de imediato para que ele possa fixar. - [3] Processo o mesmo para mudanças pequenas ou grandes features. - [12] A garantia de qualidade é feita pelo sonar (análise estática) e pelo Jenkins (tests automáticos). 	<ul style="list-style-type: none"> - [1] Utiliza o SCRUM e tem um processo bem definido. Sprint de 2 semanas. - [2] Durante a estruturação da tecnologia utilizada, não utiliza tecnologia Feature Toggle. 	<ul style="list-style-type: none"> - [30] Nunca utilizou a técnica de dark launches. - [28] Sobre tests A/B, é muito grande, então há uma certa pressa para entregar as novas funcionalidades. - [31] A equipe de desenvolvimento não tem pressa para entregar as novas funcionalidades. 	<ul style="list-style-type: none"> - [24] Sobre tests A/B, é muito grande, então há uma certa pressa para entregar as novas funcionalidades. - [25] Sobre tests A/B, é muito grande, então há uma certa pressa para entregar as novas funcionalidades. 	<ul style="list-style-type: none"> - [4] Em um horário entre na mesma hora para todos.
P4 - HW - Corp	<ul style="list-style-type: none"> - [3] Os sketches das placas e os arquivos para firmware são salvos utilizando o git todos na mesma branch. 	<ul style="list-style-type: none"> - [22] Devido a restrições da tecnologia utilizada, não utiliza tecnologia Feature Toggle. 	<ul style="list-style-type: none"> - [5] Não há uma equipe separada de HW e Infraestrutura e os times de HW e firmware trabalham em conjunto 	<ul style="list-style-type: none"> - [23] Como o projeto ainda está na fase de protótipo, não há necessidade de health checks. 	<ul style="list-style-type: none"> - [1] Processo de HW é bem pouco malável. Geralmente as coisas são decididas logo no começo, pois mudar no final é custoso. - [7] Dependendo do tamanho da mudança, é necessário ou não uma outra prova de conceito e prototipação. Outras mudanças podem não necessitar de usários de teste, isso depende do tamanho da mudança. - [11] O que pode ser automatizado é automatizado, mas devendo contento, essa parte se torna bem mais difícil. 	<ul style="list-style-type: none"> - [6] No horário das apresentações é feito um review de código para identificar possíveis erros. - [8] Com relação ao tamanho da base de usários de teste isso depende do tamanho do cliente, o PO decide se ele quer testar com todos ou com apenas uma parcela. 	<ul style="list-style-type: none"> - [24] Devido a restrições da tecnologia utilizada, não utiliza Dark Launches. - [25] Não utiliza tests A/B 	<ul style="list-style-type: none"> - [26] Sobre tests A/B, é muito grande, então há uma certa pressa para entregar as novas funcionalidades. - [27] A base de usuários é pequena para poder utilizar o processo de tests AB. 	<ul style="list-style-type: none"> - [4] Em um horário entre na mesma hora para todos.
P5 - Jogos - Startup	<ul style="list-style-type: none"> - [11] Utiliza branches no modo hub flow. Apesar de entrar no branch master, o que está preparado para ir para produção. 	<ul style="list-style-type: none"> - [23] Utiliza features toggle para escalar funcionalidades para os usuários enquanto rodam tests manuais.. - [15] Todos os desenvolvedores são devops. 	<ul style="list-style-type: none"> - [14] O responsável por monitorar as entregas é o líder técnico da equipe, que é responsável por gerenciar as funcionalidades que são necessárias para o cliente. Ele também é responsável por lidar com os problemas que surgem durante o desenvolvimento. - [24] Não tem muitos testes, é mais um pouco de planilha, depois de uma funcionalidade para garantir que ela seja suficiente para não necessitar de testes manuais. - [25] Há casos de desenvolvedores que fazem o health check apenas para garantir que a funcionalidade correладamente. 	<ul style="list-style-type: none"> - [12] O processo de entrega é bem definido, mas alguns passos dependem de funcionários. Existem funcionalidades que são necessárias para o cliente. Ele também é responsável por lidar com os problemas que surgem durante o desenvolvimento. - [16] O grupo de teste é dividido entre os usuários de teste que são responsáveis por lidar com o código e os testes que são responsáveis por lidar com os usuários de teste. 	<ul style="list-style-type: none"> - [6] Alguns features são testados em grupos de testes antes de enviar para todos os usuários. - [8] Com relação ao tamanho da base de usuários de teste isso depende do tamanho do cliente, o PO decide se ele quer testar com todos ou com apenas uma parcela. 	<ul style="list-style-type: none"> - [23] Utilizam features toggle para escoder funcionalidades para os usuários enquanto rodam tests manuais.. - [22] Não utiliza tests A/B - [27] A base de usuários é pequena para poder utilizar o processo de tests AB. 	<ul style="list-style-type: none"> - [26] Sobre tests A/B, é muito grande, então há uma certa pressa para entregar as novas funcionalidades. - [27] A base de usuários é pequena para poder utilizar o processo de tests AB. 	<ul style="list-style-type: none"> - [4] Em um horário entre na mesma hora para todos. 	

APÊNDICE B TABELA ENTREVISTADO-PRÁTICA-CÓDIGOS

Denominação	TBD	FT	AWA	HC	DOC	PIP	CAN	DAR	AB
P6 - Web - Startup	- [4] Cada feature é feita em uma branch separada. Apenas quando é concluída ela é revisada pelo gerente e colocada na branch principal.	- [12] Apresenta quando tudo é pronto o que é lançado para homologação.	- [17] O gerente que libera novas versões, mas todos do time saem como fazer isto.	- [18] Não há um time de operações.	- [26] Desenvolvedores não ficam de plantão depois de lançar uma nova funcionalidade.	- [11] Não há um processo fixo. A ideia é fazer tudo que é necessário para aquela nova entrega.	- [8] Uma mudança passa primeiro para homologação e é executada apenas por uma parte de Clientes.	- [27] Testes A/B são só utilizados seis a nove de usuários e muito pequeno.	
P7 - Web - Startup	- [5] Assim que uma feature nova entra na master, ela é entregue para o usuário, isso é baseado em commits.	- [25] Se a feature não é pronta não é entregue.	- [19] Deve-se lidar com problemas de um problema, o passo a passo para a entrega da nova versão é repassado para todos da equipe.	- [19] Tem um sistema que envia email quando existe algum problema em produção.	- [25] Saia a devolução de papéis apresentar ao time e seu supervisor.	- [11] Sobre a devolução de papéis apresentar ao time e seu supervisor.	- [11] Processo de entrega implementa Alre um PR para Review, integrar na master faz um deploy.	- [9] Não conhecia a técnica de Dark Launches.	- [27] Testes A/B são utilizados seis a nove de usuários e muito pequeno.
P8 - Web - SME	- [6] Todo o algo é integrado na master.	- [10] Toda vez que merge na master, visto que na época só é desenvolvedor.	- [26] Desenvolvedores não entregam algo feito para SRE, entregam monitoramento através do Grafana.	- [12] Não há uma definição para que a pessoa que fez a mudança faça um pouco mais depois de ela ser liberada.	- [11] Não tem um processo de feedback do cliente.	- [13] Como o entrevistado é responsável pelo monitoramento, ele é quem faz a entrega.	- [11] Não tem um processo bem estruturado.	- [4] Não utiliza partial rollouts.	- [4] Não utiliza partial rollouts.
P9 - Web - Startup	- [7] De 3 em 3 dias código é aberto para integrações com outras mas só vai para o cliente no final da sprint.	- [8] Toda vez que merge na master, visto que na época só é desenvolvedor.	- [27] Não utiliza a técnica de feature flags.	- [12] O desenvolvedor é responsável por monitorar a entrega.	- [12] O desenvolvedor com relação a feature que está sendo entregue. A pessoa que faz é a mesma que entrega.	- [12] Há um ownership com relação a desenvolvedores que ficam analisando desenvolvedores que ficam analisando.	- [11] Não tem um processo bem estruturado.	- [4] Não utiliza partial rollouts.	- [4] Não utiliza partial rollouts.
P10 - Web Front End - SME	- [8] No time de entrevistado não há deployment e integração continua.	- [9] Utiliza branches remotas para adicionar coisas novas no código.	- [28] Não utiliza a técnica de feature flags.	- [13] Existem sistemas de monitoramento próprios da toda equipe conseguindo ver através de alertas.	- [13] Existem sistemas de monitoramento próprios da toda equipe desenvolvedores que ficam analisando.	- [12] Apesar de uma feature entrar em produção, não é separado que o desenvolvedor que fez o trabalho. Não é dividido entre o dev e o ops.	- [12] As features são entram quando estão finalizadas.	- [27] Não é necessário ter uma máquina para aplicar algo do produtivo.	- [27] Não é necessário ter uma máquina para aplicar algo do produtivo.
P11 - Web Full Stack - SME	- [1] Utiliza branches individuais. Cria uma branch e abre um PR para review.	- [10] Utiliza uma branch "principal secundária".	- [12] Há um ownership com relação a feature que está sendo entregue. A pessoa que faz é a mesma que entrega.	- [13] Existem sistemas de monitoramento próprios da toda equipe desenvolvedores que ficam analisando.	- [12] Quando o commit chega na automaticamente através do google kubernetes.	- [12] O desenvolvedor não ficam de plantão depois de deploy. O plantão é só para uma prática que a empresa estima.	- [12] Não tem um processo bem estruturado.	- [27] Não é necessário ter uma máquina para aplicar algo do produtivo.	- [27] Não é necessário ter uma máquina para aplicar algo do produtivo.
P12 - Web Full Stack - SME	- [2] Utilizam trunk-based development, não só branches, por sotme, mas sim por funcionalidade. Ele merges na branch principal todo dia. Trabalham branches com a branch de master, outras não são criadas.	- [11] Utiliza feature flags.	- [12] O conhecimento é bem difundido por todo o equipo, os profissionais são bem generalistas.	- [14] Desenvolvedores não ficam de monitoramento do projeto em produção.	- [20] Desenvolvedores não ficam de plantão depois de deploy. O plantão é só para uma prática que a empresa estima.	- [20] Desenvolvedores não ficam de plantão depois de deploy. O plantão é só para uma prática que a empresa estima.	- [25] Não é necessário ter uma máquina para aplicar algo do produtivo.	- [29] Já utilizou algumas vezes com dark launches.	- [29] Já utilizou algumas vezes com dark launches.
P13 - Web Full Stack - SME	- [3] Utilizam Feature Toggle, Out.	- [13] O entrevistado trabalha como "donor", ou seja, é o responsável pelos processos manuais e automatizados.	- [15] Utilizam o mesmo processo. Coisas que são feitas em produção.	- [21] Não há processo de monitoramento do projeto em produção.	- [21] O desenvolvedor que é responsável que fez o merge tem que é responsável que fez o merge.	- [21] O desenvolvedor que é responsável que fez o merge tem que é responsável que fez o merge.	- [25] Não é necessário ter uma máquina para aplicar algo do produtivo.	- [26] Não utiliza Tests A/B para testar se o novo sistema que está sendo implementado é útil.	- [26] Não utiliza Tests A/B para testar se o novo sistema que está sendo implementado é útil.
P14 - Web Full Stack - SME	- [4] Utilizam Dala Dog como forma de Health check, mas não conhece muito bem a respeito.	- [14] Há desenvolvedores que não se sentem confortáveis em mixer com atividades devops.	- [17] Utilizam Dala Dog como forma de Health check, mas não conhece muito bem a respeito.	- [17] O time tem um processo bem definido.	- [17] Testes automáticos rodam a cada commit e após merge é feito um novo build.	- [17] O time tem um processo bem definido.	- [24] Nunca utilizou dark launches.	- [22] Não utiliza o processo de teste A/B. Não tinha chegado num momento que fosse necessário.	- [24] Nunca utilizou tests A/B para testar se o novo sistema que está sendo implementado é útil.
P15 - Web Full Stack - SME	- [5] Passa pelo mesmo processo para bugs pequenos ou features grandes.	- [15] Passa pelo mesmo processo para bugs pequenos ou features grandes.	- [16] Todos os usuários recebem as mudanças ao mesmo tempo.	- [16] Os sites saem da opinião só o time review os mesmos.	- [26] Não utiliza tests A/B para testar se o novo sistema que está sendo implementado é útil.	- [26] Não utiliza tests A/B para testar se o novo sistema que está sendo implementado é útil.	- [24] Nunca utilizou tests A/B para testar se o novo sistema que está sendo implementado é útil.	- [24] Nunca utilizou tests A/B para testar se o novo sistema que está sendo implementado é útil.	- [24] Nunca utilizou tests A/B para testar se o novo sistema que está sendo implementado é útil.

Denominação	Code review	Testes automatizados
P1 - Web - SME	- [14] Não há code review majoritariamente pois só há ele na equipe e somente ele na empresa conhece de ruby.	- [15] Para melhorar QA: Deveria adicionar testes automatizados - [18] Problemas são comuns no sistema
P2 - Web - Corp	- [17] Code Review serve para revisar se a feature foi realmente implementada, checar formas melhores de fazer e melhorar o compartilhamento de informações dentro da equipe	- [4] Suite de testes automática que simula merge + teste no PR - [18] BDD é utilizado para automação de testes - [19] Contudo, a automação não resolve todos os problemas
P3 - Web - Corp	- [17] A entrevistada não tem muita experiência com código por ser razoavelmente nova na área, e vê a revisão de código como bastante agregadora de conhecimento. - [18] O code review dissemina conhecimento de diferentes áreas do sistema para toda a equipe. - [19] A equipe também é bem aberta para o debate e está por dentro do processo de code review e está engajada.	- [12] A garantia de qualidade é feita pelo sonar(análise estática) e pelo jenkins(testes automáticos)
P4 - HW - Corp	N/A	- [9] Há testes de validação que podem ser automáticos. - [10] Para ser automático, é criado uma Giga de testes, que é basicamente um outro hw que testa o produto final. Isso geralmente é enviado para a fabricante das placas para testar cada uma das placas prototipadas e fabricadas. - [11] O que pode ser automatizado eles automatizam, mas, devido ao contexto, essa parte se torna bem mais difícil.
P5 - Jogos - Startup	- [12] Utiliza code review.	- [10] Ele acha o processo de entrega bom, dá certo, mas acredita que seria bom ter testes mais automatizados, só que o tempo da sprint é curto.
P6 - Web - Startup	- [2] Depois do commit o gerente analisa o código. - [21] A equipe é pequena (não há uma quantidade grande de commits por dia) e o gerente cuida do código. Ele revisa todo dia nas branches de cada um dos devs e passa as devidas críticas sobre como fazer melhor.	- [20] Há testes automatizados mas rodam na máquina local antes de subir para homologação.

APÊNDICE B TABELA ENTREVISTADO-PRATICA-CODIGOS

36

Denominação	Code review	Testes automatizados
P7 - Web - Startup	<ul style="list-style-type: none"> - [16] O Code review é importante para achar erros que o criador não viu e impor melhores práticas - [17] O processo de análise de qualidade funciona mais ou menos, depende do humor dos revisores. 	<ul style="list-style-type: none"> - [2] Há testes automatizados para garantir que alguns cenários estão funcionando. - [15] Garantia da qualidade é feita através dos testes automáticos, que contém checks de Lint, cenários, etc.
P8 - Web - SME	<ul style="list-style-type: none"> - [16] Code-review não existe pois faltam pessoas qualificadas para fazer isso. 	<ul style="list-style-type: none"> - [20] Não é comum a abertura de bugs em produção, mas isso se deve ao fato de o software não ser testado tão bem. - [23] Mais testes manuais e automáticos poderiam diminuir a quantidade de bugs.
P9 - Web - Startup	<ul style="list-style-type: none"> - [18] Code review serve para receber novas perspectivas e visões de outras pessoas, assim como refinar o código e discutir decisões arquiteturais. 	<ul style="list-style-type: none"> - [15] A garantia da qualidade é feita apenas pelo review. Não há testes automatizados. - [16] As coisas são testadas manualmente em um ambiente de staging. - [24] Se houvesse uma suíte mais estruturada de teste, o número de erros em "run time" seriam diminuídos, mas o processo de entrega seria bem mais devagar.
P10 - Web Front End - SME	<ul style="list-style-type: none"> - [8] Acha o processo de entrega bom pois gera muito aprendizado, mas acha que para seniors talvez seja um pouco demorado, visto que estes têm que esperar uma revisão de código que, para a entrevistada, talvez não fizesse sentido. - [13] O code review é a única tática para não gerar muitos bugs na aplicação. tentam também deixar os prs bem pequenos. - [14] Como eles utilizam uma formatação de código montada pela própria empresa para manter o código limpo, isto é checado durante o review. - [15] O code review aumenta o aprendizado dela como engenheira junior 	<ul style="list-style-type: none"> - [5] O processo de rodar os testes também é automatizado. - [16] Gosta bastante do controle de qualidade da empresa.
P11 - Web Full Stack - SME	<ul style="list-style-type: none"> - [18] Acha que Code review não deve ser feito para comentar apenas sobre estilo de código, não compensa o esforço. - [19] Utiliza code review principalmente pela troca de conhecimento. 	<ul style="list-style-type: none"> - [20] Sente que tem muita regressão, talvez devido a uma cobertura de testes pequena. Outro fator que aumenta a quantidade de bugs é a comunicação ruim entre os times (QA e dev).

APÊNDICE C

Super Categorias

APÊNDICE C SUPER CATEGORIAS

38

Categories (general)	Categories	Codes
TBD	Utiliza a metodologia hubflow (Git Flow no Github)	P2 - [3] Utilizam a metodologia de hub flow. P5 - [11] Utiliza branches no modo hub flow. Apenas entra na branch master aquilo que está preparado para ir para produção.
		Utiliza a metodologia hubflow (Git Flow no Github) P1 - [3] Lança no fim de cada sprint (2 semanas) - [5] Entrega código na master e para os clientes a cada sprint"
	Integra o código na branch principal apenas no final da sprint	P3 - [1] Utiliza o SCRUM e tem um processo bem definido. Sprint de 2 semanas - [4] O código de cada sprint é colocado na branch principal apenas no final da mesma. P6 - [4] Cada feature é feito em uma branch separada. Apenas quando concluída ela é revisada pelo gerente e colocada na branch principal.
		P2 - [25] Há o conceito de funcionalidades como épico, que se prolongam por múltiplas sprints, mas não há o conceito de FT: o código é mergeado apenas após a feature estar completa. P3 - [6] Tem a ideia de fazer uma funcionalidade entre duas sprints, mas nunca entregam pela metade, é sempre apenas quando ela estiver completa.
FT	Utiliza o conceito de épicos, mas o código só é integrado quando terminado	Utiliza o conceito de épicos, mas o código só é integrado quando terminado P1 - [20] Se a feature não está pronta, ela não entra em produção P6 - [12] Apenas quando tiver tudo pronto é que é lançado para homologação P7 - [25] Se a feature não tá pronta não entrega. Não utiliza condicionais para esconder código não pronto P9 - [27] As features só entram quando estão finalizadas P11 - [25] Quando uma funcionalidade não está pronta ela não é entregue, não utiliza Feature Toggle. Ou entrega quando está pronto ou particiona o código e adiciona pequenas partes até ter a funcionalidade de fato.
	A feature só vai para o ambiente de produção quando concluída	P1 - [8] É ele quem faz a entrega nos ambientes - [10] Há apenas ele na equipe, e ele é responsável pelo deploy e manutenção dos ambientes. P2 - [8] Existe uma ferramenta de CLI do projeto que automatiza a criação de algumas coisas e facilita o provisionamento de ambientes. - [9] Há a cada sprint o disparo de um release notes para toda a equipe e uma versão simplificada para os usuários do sistema interessados nas novas versões. - [13] Não existe o papel de Devops, há um revezamento entre todos da equipe para que o conhecimento seja difundido P5 - [14] O responsável por monitorar as entregas é o líder técnico da equipe. Todos conseguem deployar, sabem como fazer, mas geralmente quem faz é o líder - [15] Todos os desenvolvedores são devops.
	O time de desenvolvimento é o mesmo que faz deploy e mantém a aplicação rodando. Não há um time separado de operações	P6 - [17] O gerente que libera novas versões, mas todos do time sabem como fazer isto. - [18] Não há um time de operações. - [19] Depois de um problema, o passo a passo para a entrega de novas versões foi repassado para todos da equipe. P8 - [11] O time de desenvolvimento é o mesmo que faz deploy e recebe o feedback do cliente. - [12] O desenvolvedor é responsável por monitorar a entrega. P9 - [12] Há um ownership com relação a feature que está sendo entregue. A pessoa que faz é a mesma que entrega. Todos do time sabem fazer tudo: "Não tem divisão entre o dev e o ops".
	Há um time de Devops específico, mas não há grandes silos entre este e a equipe de desenvolvimento	P3 - [7] Quando há uma nova versão no ar é enviado um email para todos os interessados. - [9] Há uma equipe dedicada para devops. Mas sempre temos uma conversa entre a equipe do projeto e o SRE para melhorar o produto final. - [11] O deploy é feito pelo projeto, não por SRE P7 - [11] Sobre a divisão de papéis, apesar de ter um time específico de SRE, os desenvolvedores também são devops. Qualquer pessoa do time pode entregar uma nova versão. - [14] Os desenvolvedores não entregam só algo feito para SRE entregar, eles também entendem como funciona a infraestrutura.

HC	Há apenas poucas verificações não muito complexas a respeito do sistema de produção	P1 - [17] Não há HC rodando atualmente, mas já houve (está quebrado). Há apenas um check sobre o certificado https. P5 - [24] Não tem muitos testes automáticos de health check, apenas um caso para testar se o servidor do jogo está rodando.
DOC	Não é uma prática definida, mas acontece implicitamente pela equipe	P7 - [12] Não há uma definição para que a pessoa que fez a mudança fique um pouco mais depois de ela ser liberada, mas isso acontece implicitamente. P8 - [13] Como o entrevistado é responsável pela entrega, algumas vezes foi necessário que a gerente conversasse com ele fora do expediente para ajeitar algo de produção. - [21] Após uma nova feature entrar em produção, não é esperado que o desenvolvedor fique de plantão, mas já aconteceram alguns casos. O entrevistado procura estar disponível. P9 - [23] Em alguns casos há desenvolvedores que ficam analisando as métricas e o sistema depois de um deploy em produção, mas depende da mudança.
PIP	Há um processo bem definido, mas este contém poucos ou nenhum passos automatizados	P1 - [1] O ciclo consiste em: Funcionalidade nova, testes manuais em staging e deploy - [6] Consegue fazer a entrega de forma rápida (menos de 30 minutos), mas não tem um processo automatizado - [7] Ele acredita que poderia ter um processo mais automatizado - [12] Existe um processo estagiado mas não automatizado P4 - [1] Processo de HW é bem pouco maleável. Geralmente as coisas são decididas logo no começo, pois mudar no final é custoso. - [7] Dependendo do tamanho da mudança, é necessário ou não uma outra prova de conceito e prototipação. Outras mudanças podem não passar por todas as fases. O time decide sobre o tamanho da mudança. - [11] O que pode ser automatizado eles automatizam, mas, devido ao contexto, essa parte se torna bem mais difícil.
	Não tem um processo bem estruturado e não há passos automatizados.	P6 - [11] Não há um processo fixo. A ideia é fazer tudo que tem que ser feito para aquela nova entrega. - [13] Não há passos de deploy automatizados. P8 - [1] Não tem um processo bem estruturado. - [2] Após o commit e o review das features na sprint review, as mudanças são enviadas para produção. Não tem nada automatizado.
	Tem um processo estruturado e passos automatizados, mas este não é seguido para todo tipo de mudança.	P5 - [2] O processo de entrega é bem definido, mas alguns passos dependem da funcionalidade. Existem funcionalidades que são testadas manualmente para validação. Mas tem algumas outras que são pequenas o suficiente para não necessitar de testes manuais. - [3] A equipe define o tamanho da mudança e que pipeline ela vai seguir. P10 - [2] Quando o commit chega na master o deploy é feito automaticamente através do google kubernetes. - [3] Coisas pequenas ou grandes não utilizam o mesmo processo. Coisas pequenas seguem um caminho "mais curto" para chegar em produção. - [4] O processo de rodar os testes também é automatizado.

CAN	Utiliza um ambiente de homologação apenas para testes e validação de requisitos, mas este utiliza um banco de dados diferente do de produção	P1 - [2] A funcionalidade é lançada para todos que forem afetados, já que o código é baseado na linha de produto - [4] Em um hotfix entra na mesma hora para todos - [13] Existe um ambiente de staging parecido com produção, mas este é utilizado apenas para testes da equipe interna e apresentações
		P2 - [5] Depois que é gerada a release, ela é entregue no ambiente de homologação para checks de regressão e de visualização inicial do cliente. Ao fim da sprint, essa release é levada para produção
DAR	As features são sempre entregues para todos os usuários ao mesmo tempo	P6 - [8] Uma mudança passa primeiro para homologação e é testada apenas por uma parte dos usuários (que são os clientes). - [9] Homologação tem um banco de dados diferente de produção
		P3 - [5] Uma feature é sempre lançada para todos os usuários
AB	Não conhecia a técnica de dark launches	P8 - [4] As features são enviadas para todos os usuários de uma só vez.
		P11 - [6] Todos os usuários recebem as mudanças ao mesmo tempo
DAR	Nunca utilizou a técnica de dark launches	P2 - [26] Não conhecia a técnica de dark launches
		P6 - [28] Não conhecia a técnica de Dark Launches
AB	Não utiliza A/B pela baixa quantidade de usuários	P8 - [27] Não conhecia e não utiliza a técnica de dark launches
		Não conhecia a técnica de dark launches
AB	Testes A/B não se aplicavam ao contexto da aplicação	P1 - [22] Conhece a técnica de dark launches, mas nunca fez uso dela.
		P10 - [24] Nunca utilizou dark launches.
Code Review	Acredita que o code-review funciona para troca de conhecimento	P3 - [24] Não utilizou a técnica de dark launches
		P5 - [26] Não utilizava tests A/B. - [27] A base de usuários é pequena para poder utilizar o processo de Testes A/B
Testes automáticos	Acredita que o aumento da cobertura de testes automáticos pode diminuir a quantidade de bugs em produção	P6 - [27] Testes A/B não são utilizados pois a base de usuários é muito pequena.
		P11 - [24] Não utiliza tests a/b pois a base de usuários é pequena
Testes automáticos	Acredita que o aumento da cobertura de testes automáticos pode diminuir a quantidade de bugs em produção	P7 - [4] Não utiliza partial rollouts. - [24] Testes A/B não são utilizados por achar que não se aplica ao contexto da aplicação
		P9 - [26] Não utiliza Testes A/B pois é difícil aplicar nos serviços que usam, visto que são sistemas de uso interno
Code Review	Acredita que o code-review funciona para troca de conhecimento	P3 - [17] A entrevistada não tem muita experiência com código por ser razoavelmente nova na área, e vê a revisão de código como bastante agregadora de conhecimento. - [18] O code review dissemina conhecimento de diferentes áreas do sistema para toda a equipe.
		P10 - [15] O code review aumenta o aprendizado dela como engenheira junior
Testes automáticos	Acredita que o aumento da cobertura de testes automáticos pode diminuir a quantidade de bugs em produção	P11 - [19] Utiliza code review principalmente pela troca de conhecimento.
		P1 - [15] Para melhorar QA: Deveria adicionar testes automatizados
Testes automáticos	Acredita que o aumento da cobertura de testes automáticos pode diminuir a quantidade de bugs em produção	P5 - [10] Ele acha o processo de entrega bom, dá certo, mas acredita que seria bom ter testes mais automatizados, só que o tempo da sprint é curto.
		P8 - [23] Mais testes manuais e automáticos poderiam diminuir a quantidade de bugs
Testes automáticos	Acredita que o aumento da cobertura de testes automáticos pode diminuir a quantidade de bugs em produção	P9 - [24] Se houvesse uma suíte mais estruturada de teste, o número de erros em "run time" seriam diminuídos, mas o processo de entrega seria bem mais devagar.
		P11 - [20] Sente que tem muita regressão, talvez devido a uma cobertura de testes pequena. Outro fator que aumenta a quantidade de bugs é a comunicação ruim entre os times (QA e dev).

Referências Bibliográficas

- [1] 14th annual state of agile report. <https://stateofagile.com/#ufh-i-615706098-14th-annual-state-of-agile-report/7027494>. Acessado em 23/04/2021.
- [2] Devops. <https://pt.wikipedia.org/wiki/DevOps>. Acessado em 20/04/2021.
- [3] O que é o porto digital. <https://www.portodigital.org/parque/o-que-e-o-porto-digital>. Acessado em 07/09/2021.
- [4] John Allspaw and Paul Hammond. 10+ deploys per day: Dev and ops cooperation at flickr. 2009.
- [5] Bacchelli et al. Expectations, outcomes, and challenges of modern code review. 2013.
- [6] Carolina Cozer. O que é scrum e como aplicá-lo em startups? <https://www.whow.com.br/insights/o-que-e-scrum-e-como-aplica-lo-em-startups/>. Acessado em 24/04/2021.
- [7] Adam Debbiche et al. Challenges when adopting continuous integration: A case study. In *International Conference on Product-Focused Software Process Improvement*, pages 17–32. Springer, 2014.
- [8] Abhishek DWARAKI et al. Gitflow: Flow revision management for software-defined networks. 2015.
- [9] D. G. Feitelson et al. Development and deployment at facebook. *IEEE Internet Computing*, 17(4):8-17:8–17, 2013.
- [10] Wagner Felidré et al. Continuous integration theater. In *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–10. IEEE, 2019.
- [11] D. L. B. FILHO. Experiências com desenvolvimento ágil. *Instituto de Matemática e Estatística da Universidade de São Paulo (Dissertação de Mestrado)*, 2008.
- [12] Martin Fowler. Continuous delivery. <https://martinfowler.com/bliki/ContinuousDelivery.html>. Acessado em 20/04/2021.
- [13] Martin Fowler. Continuous integration. <https://martinfowler.com/articles/continuousIntegration.html>. Acessado em 20/04/2021.

- [14] Barney G. Glaser. Basics of grounded theory analysis: Emergence vs. forcing. 1992.
- [15] J. Highsmith and A. Cockburn. Agile software development: the business of innovation. *Compute*, 34(9):120–127, 2001.
- [16] Michael Hilton et al. Continuous integration (ci) needs and wishes for developers of proprietary code. 2016.
- [17] Pete Hodgson. Feature toggles. <https://martinfowler.com/articles/feature-toggles.html>. Acessado em 17/04/2021.
- [18] Jez Humble and David Farley. *Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation*. Addison-Wesley Professional, 2010.
- [19] Ron Kohavi et al. Practical guide to controlled experiments on the web: Listen to your customers not to the hippo. pages 956–967, Agosto 2007.
- [20] Ingo Weber Len Bass and Liming Zhu. *DevOps: A Software Architect's Perspective*. Addison-Wesley Professional, 2015.
- [21] John Micco. Tools for continuous integration at google scale. https://www.youtube.com/watch?v=KH2_sB1A6lA. Acessado em 20/04/2021.
- [22] Akond Ashfaque Ur Rahman et al. Synthesizing continuous deployment practices used in software development. 2015.
- [23] Gerald Schermann et al. An empirical study on principles and practices of continuous delivery and deployment. 2016.
- [24] Ken Schwaber and Mike Beedle. *Agile Software Development with SCRUM*. Prentice-Hall, 2002.
- [25] Mitchel Douglas Tony Savor and Michael Gentili. Continuous deployment at facebook and oanda. 2016.
- [26] Gustavo Pinto e Rodrigo Bonifácio Welder Pinheiro Luz. Adopting devops in the real world: A theory, a model, and a case study. *Journal of Systems and Software*, 157, 2019.