Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Modernização do Feature-Trace mediante a adoção de princípios DevOps

Helio Adson Oliveira Bernardo

Monografia apresentada como requisito parcial
para conclusão do Curso de Engenharia da Computação

Orientador
Prof.a Dr.a Genaína Rodrigues

Brasília
2021

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Modernização do Feature-Trace mediante a adoção de princípios DevOps

Helio Adson Oliveira Bernardo

Monografia apresentada como requisito parcial
para conclusão do Curso de Engenharia da Computação

Prof.a Dr.a Genaína Rodrigues (Orientador)
CIC/UnB

Prof. Rafael Fazzolino Pinto Barbosa
FGA/UnB

Prof. Dr. Rodrigo Bonifácio de Almeida
CIC/UnB

do Curso de Engenharia da Computação

Brasília, 05 de novembro de 2021

Dedicatória

Primeiramente dedico esse trabalho à minha mãe, Sandra Correia de Oliveira, que sempre proporcionou condições para que eu seguisse em frente com os estudos.

Dedico também à minha parceira, Dayana Maia. Sem seu apoio eu não teria chegado tão longe.

Por fim dedico este trabalho aos demais familiares, em especial meu irmão, Heliudson de Oliveira Bernardo, e para todos os meus amigos que me apoiaram nessa jornada, principalmente os grupos *jogadores* e *vencedores*.

Agradecimentos

Agradeço a todos os familiares e amigos que acreditaram na minha pessoa. Agradeço também à instituição Universidade de Brasília, que possibilitou uma grande mudança e avanço em minha vida. À Vanessa Nunes, pelo apoio no desenvolvimento deste trabalho.

Por fim, agradeço à minha orientadora Prof.a Dr.a Genaína Nunes Rodrigues pelo apoio e oportunidade.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES), por meio do Acesso ao Portal de Periódicos.

Resumo

A ferramenta Feature-Trace é utilizada para a obtenção de métricas de um *software* analisado a partir de rastreabilidade oferecida por artefatos presentes no código fonte do projeto analisado, principalmente pelo BDD. Com a obtenção de métricas, em especial o Perfil Operacional, o Feature-Trace possibilita a priorização de esforço em um *software*, principalmente para a seleção de casos de teste. No entanto, a ferramenta apresenta necessidades de evolução em sua estrutura, usabilidade e configurabilidade a fim de simplificar o uso deliberado por parte de usuários, além de facilitar o desenvolvimento de novas funcionalidades. Apresenta-se neste trabalho uma solução para estes problemas em forma da adoção de conceitos de *DevOps*. Buscou-se aprimoramento no fluxo de desenvolvimento de contribuições para que este apresente artefatos que aumentam a agilidade e qualidade na integração de novas funcionalidades. Almejou-se também uma nova abordagem para o fluxo de uso da ferramenta, simplificando-o e removendo dificuldades. A validação de contribuições em relação a usabilidade e configurabilidade do Feature-Trace deu-se através de um estudo de caso, manifestado em forma de sessão prática com desenvolvedores. Os resultados obtidos evidenciaram um alinhamento com técnicas e práticas amplamente utilizadas pela indústria e comunidade de desenvolvimento de *software*, manifestado através das contribuições que compõem a adoção de *DevOps*.

Palavras-chave: Feature-Trace, BDD, Perfil Operacional, DevOps, Configurabilidade de Software, Usabilidade de Software, Python

Abstract

The Feature-Trace tool is used to obtain metrics of an analyzed *software* from the traceability offered by artifacts present in the source code of the analyzed project, mainly by BDD. By obtaining metrics, especially the Operational Profile, Feature-Trace enables the prioritization of effort in a *software*, mainly for the selection of test cases. However, the tool needs to evolve in its structure, usability and configurability in order to simplify the deliberate use by users, in addition to facilitating the development of new features. This work presents a solution to these problems in the form of adopting *DevOps* concepts. We sought to improve the development flow of contributions so that it presents artifacts that help with the agility and quality in the integration of new features. A new approach to the flow of tool use was also sought, simplifying it and removing difficulties. The validation of contributions to the usability and configurability of Feature-Trace took place through a case study, expressed in the form of practical session with developers. The results obtained showed an alignment with techniques and practices widely used by the industry and the *software* development community, manifested through the contributions that make up the adoption of *DevOps*.

Keywords: Feature-Trace, BDD, Operational Profile, DevOps, Software Configurability, Software Usability, Python

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Obter métricas e informações sobre um projeto desenvolvido é algo comum ao desenvolvimento de *software*. Diversas abordagens para análise de *softwares* foram elaboradas até então por inúmeros pesquisadores e desenvolvedores que sempre almejam otimizar esse processo, tornando-o menos complexo e aumentando seu impacto. De acordo com as inúmeras soluções e metodologias elaboradas com este intuito, a relevância de novas abordagens para a comunidade de desenvolvimento de *software* requer um certo grau de aptidão, principalmente para ferramentas desenvolvidas com este propósito. Dado este contexto, este trabalho concentra-se em contribuir com a relevância de uma ferramenta de análise de obtenção de métricas de um software, refletindo sobre suas esferas de desenvolvimento e de usabilidade pelo usuário final. Essa contribuição se dá através da adoção de princípios *DevOps*, termo associado a um conjunto de técnicas e práticas de desenvolvimento ágil aplicadas a fim de otimizar o ritmo de desenvolvimento e qualidade de aplicações.

Serão descritos nas próximas sessões os problemas que encorajaram esta pesquisa, assim como os resultados almejados e hipótese, esta que constitui as contribuições em forma de proposta de solução para estas dificuldades.

1.1 Problema

A ferramenta Feature-Trace, desenvolvida por pesquisadores do Departamento de Ciência da Computação da Universidade de Brasília, tem como objetivo a obtenção de métricas de um *software* analisado a partir de rastreabilidade oferecida por artefatos presentes no código fonte do projeto analisado [1].

O artefato que apresenta rastreabilidade que é foco do Feature-Trace se manifesta através de testes BDD (Behaviour Driven Development), uma abordagem de desenvolvimento de *software* que visa a aproximação de detentores do produto, usuários e desenvolvedores

através da elaboração da descrição de funcionalidades do produto em linguagem abstrata, próxima à humana [2]. Testes de aceitação são desenvolvidos a partir dessas descrições de funcionalidades, construindo um processo que apresenta rastreabilidade em relação ao *software*.

O Feature-Trace mostra-se com uma ferramenta útil e poderosa ao oferecer redução de esforço para a obtenção de métricas de um *software*, auxiliando na priorização e seleção de casos de testes em um *software*. Porém, a mesma ainda sofre com limitações em sua estrutura, usabilidade e configurabilidade, encontrando-se em uma versão inicial, um estado restrito e que apresenta grandes oportunidades para melhoria. O potencial da ferramenta e o espaço para melhorias fomentam este trabalho.

Configurar e utilizar o Feature-Trace não é algo trivial. Sua arquitetura é dividida entre dois serviços distintos: cliente e servidor. Para configuração e utilização da ferramenta, é necessária a obtenção do código fonte de cada um desses serviços em conjunto com suas instalações e configurações. Esse processo requer muito esforço e remove o foco do *software* a ser analisado pelo Feature-Trace.

Além da perspectiva do usuário, este estado bruto da ferramenta dificulta o desenvolvimento de novas funcionalidades e trabalhos futuros. Com o código fonte dos serviços restritos a máquinas de usuários, novas versões do Feature-Trace não alcançam estes de forma natural, dificultando a entrega de valor da ferramenta.

O *Deployment* (entrega de novas versões de um *software*) para este caso restringe-se a disponibilização do código fonte atualizado em um repositório virtual, algo não ideal para este tipo de ferramenta.

Por fim, nenhuma técnica ou prática que auxilia o desenvolvimento de contribuições mostra-se presente. Como o Feature-Trace se trata de um projeto de código aberto, o ambiente não receptivo dificulta o fluxo de desenvolvimento de novas funcionalidades.

Todas essas limitações restringem o uso deliberado do Feature-Trace e também sua continuidade, quebrando o ciclo de vida do projeto, dado os impedimentos para com seu desenvolvimento. Mesmo com grande potencial, a ferramenta apresenta necessidades de evolução como projeto de software tanto na perspectiva do usuário final quanto na do desenvolvimento de contribuições.

Em suma, o problema do Featrure Trace é que este possui gargalos para sua configurabilidade, usabilidade e desenvolvimento de novas funcionalidades. Seu potencial está oculto devido ao estado bruto no qual a ferramenta encontra-se, impactando negativamente a perspectiva de usuário e desenvolvimento.

1.2 Hipótese

No intuito de solucionar estas limitações, planeja-se uma adoção de princípios e práticas de *DevOps* para com o Feature-Trace. Essa adoção mostra-se adequada como solução para os problemas identificados, já que as técnicas associadas ao *DevOps* buscam otimizar o desenvolvimento de *software*, impactando a elaboração de novas funcionalidades. Além disso, essas práticas também oferecem aprimoramento na entrega de valor e encapsulamento de uma aplicação, contribuindo para com a perspectiva do usuário final.

Sendo assim, espera-se que esta adoção resulte em benefícios ao usuário, reduzindo o esforço para sua configuração e utilização, além de estruturar e aprimorar o fluxo de desenvolvimento, alinhando a ferramenta com práticas e técnicas modernas.

1.3 Objetivo geral

O objetivo principal deste trabalho é aprimorar a usabilidade de configurabilidade da ferramenta e tornar o Feature-Trace suscetível a contribuições através da adoção de princípios de *DevOps*, selecionados mediante a pertinência de cada um para com o contexto da ferramenta. Sendo assim, este trabalho propõe impactar positivamente tanto o panorama do usuário quanto o de desenvolvimento.

1.4 Objetivos específicos

Para alcançar o objetivo geral deste trabalho, definiu-se os seguintes objetivos específicos:

- Tornar ágil e usual o processo de desenvolvimento de novas funcionalidades para o Feature-Trace.
- Possibilitar que novas contribuições ao Feature-Trace possam alcançar o usuário de forma natural e sem gargalos.
- Facilitar a configurabilidade e usabilidade da ferramenta, tornando-a utilizável no cotidiano de um desenvolvedor e diminuindo o grau de esforço para o uso do Feature-Trace.
- Alinhar o Feature-Trace a técnicas e conceitos utilizados pela indústria e comunidade de desenvolvimento de *software*.

Cada um desses objetivos tem correlação com a adoção de conceitos de *DevOps*, motivando aplicação de conceitos remetentes a este termo.

1.5 Organização do trabalho

Os demais capítulos deste trabalho estão dispostos da seguinte forma:

No Capítulo 2, são abordados os tópicos necessários para o entendimento deste trabalho. Os materiais citados neste capítulo formam a base de conhecimento desta pesquisa e são imprescindíveis para sua construção.

Para o Capítulo 3, detalha-se a execução da adoção de *DevOps* para o Feature-Trace. Apresenta-se uma lista de conceitos e práticas que compõem as contribuições deste trabalho, aplicadas através do uso de uma metodologia proposta que aborda a adoção de *DevOps*. Cada uma dessas contribuições são correlacionadas com sua motivação e resultado esperado de acordo com os objetivos almejados.

No Capítulo 4, mostra-se a validação das contribuições deste trabalho através de um estudo de caso na forma de sessão prática com desenvolvedores, onde estes utilizam a ferramenta e apresentam opiniões úteis como métrica sobre a mesma. O foco deste estudo é a verificação de contribuições para com a configurabilidade e usabilidade do Feature-Trace.

Por fim, o capítulo 5 aborda a conclusão deste trabalho.

Capítulo 2

Fundamentação Teórica

Nas próximas seções serão descritos tanto os tópicos necessários para o entendimento deste trabalho como os métodos, os materiais e as ferramentas que foram utilizados e que formam a base de conhecimento desta pesquisa.

2.1 BDD

Behaviour Driven Development (BDD) [2] é uma metodologia de desenvolvimento de software derivada de outras técnicas como o *Test Driven Development* (TDD) [3], que tem como contexto utilizar testes de aceitação em linguagem próxima à humana, ou seja, mais abstrata que linguagens de programação em geral. Esses testes são escritos como *Behavior Tests* que, como o próprio nome expressa, testam o comportamento de uma aplicação. O uso de uma linguagem com alta legibilidade tem como objetivo aproximar os *stakeholders* (indivíduos e organizações impactados pelo projeto) ao levantamento de requisitos, além de processos de validação e verificação do projeto.

Dentre todas as linguagens usadas para o desenvolvimento do BDD destaca-se o *Gherkin* [4], que proporciona uma série de regras de sintaxe para descrição de funcionalidades. Arquivos *.feature* são escritos em conjunto com o cliente do projeto detalhando cada *feature* (funcionalidade) com seus *scenarios* (cenários) e *steps* (passos). O seguinte trecho de código em linguagem *Gherkin* demonstra a descrição de uma *feature* com dois *scenarios*:

```
Funcionalidade: Adivinhe a palavra
  # Primeiro exemplo de cenário com dois passos
  Cenário: Criador inicia o jogo
    Quando o Criador inicia o jogo
    O Criador espera um Adivinhador se juntar ao jogo
```

```
# Segundo exemplo de cenário com três passos
Cenário: Adivinhador se junta ao jogo
  Dado que o Criador começou o jogo com a palavra "sedoso"
  Quando o Adivinhador se juntar ao jogo do Criador
  Então o Adivinhador precisa adivinhar uma palavra com 5 letras
```

Onde *Dado*, *Quando* e *Então* são regras de sintaxe disponibilizadas pelo *Gherkin* para descrição dos *steps* de uma funcionalidade. Usando esse conjunto de regras pode-se criar os testes de comportamento com legibilidade alta e deixá-los próximo da linguagem natural.

Os arquivos *.feature* são usados para a elaboração de testes automatizados e servem como documentação de funcionalidades do projeto. Esses testes automatizados são criados com expressões regulares com base nas bibliotecas que analisam os testes, como no exemplo a seguir que utiliza o *cucumber* para automação do código em linguagem *Gherkin*:

```
Dado /^o Adivinhador inicia o jogo com a palavra "([^"]*)"$/ do |palavra|
    Jogo.create(palavra)
end
```

Os testes de comportamento, quando analisados de forma automática, garantem que a aplicação faz sentido e está alinhada com os requisitos especificados pelos *stakeholders*. Essa confiabilidade no que está sendo desenvolvido no projeto do ponto de vista de requisitos não é alcançável apenas com uma cobertura de outros tipos de testes. Além disso, de acordo com [5], o BDD possibilita um foco maior do time na identificação e entendimento da importância de funcionalidades a nível de produto.

A ferramenta alvo de contribuições deste trabalho, o Feature-Trace, utiliza a rastreabilidade de *software* oferecida pelo BDD para extrair métricas, sendo que mais detalhes sobre esse tópico serão abordados nas próximas seções.

2.2 Perfil Operacional

Outro tópico necessário para entender este trabalho, o Perfil Operacional de um Software, de acordo com John Musa [6], é a quantificação de ocorrência de cada uma de suas entidades. Trata-se de um conjunto de operações que um software efetua associado às suas respectivas probabilidades. Com esta métrica, é possível obter a importância de cada entidade e seu impacto na confiabilidade do *software*.

Musa [6] ainda cita que existem 5 etapas para o desenvolvimento do perfil operacional de um sistema:

- Identificação do perfil do cliente, elaborando e tomando noção do grupo de usuários do sistema;
- Elaboração do perfil de usuário, composto por pessoas que operam o sistema;
- Concepção do perfil do modo de sistema, definindo as formas distintas de operação do sistema e como os usuários irão utilizá-lo;
- Formação do perfil funcional, que a probabilidade de uma funcionalidade específica ser executada; e
- Elaboração do perfil operacional, determinando a probabilidade de uma operação ocorrer, sendo que essas operações são o que compõem as funcionalidades do sistema.

A definição do perfil operacional traz discernimento sobre o sistema, reduzindo custos com operações com impacto mínimo no projeto. Além disso, de acordo com [6], guia a distribuição de esforço entre escopos distintos do projeto, como os requisitos, *design* e o código em si. Dentre essas vantagens do uso do perfil operacional apresentadas por Musa, destaca-se que o conhecimento sobre as operações faz com que o processo de teste do sistema seja facilitado, contribuindo na distribuição otimizada dos mesmos. O PO (perfil operacional) é uma métrica alvo do Feature-Trace, ferramenta para análise de *software* que é apresentada a seguir.

2.3 Feature-Trace

O *Feature-Trace* é uma ferramenta que utiliza a rastreabilidade oferecido pelo *BDD* para levantar métricas de um *software* a ser testado, principalmente seu perfil operacional [1]. A obtenção semi-automática do perfil operacional permite a priorização de testes de forma otimizada, minimizando o esforço com estratégias e passos usuais na elaboração do mesmo. O Feature-Trace é foco e alvo das contribuições deste trabalho, sendo assim o entendimento sobre seu funcionamento e arquitetura é essencial e será abordado nesta seção.

Além do perfil operacional, a ferramenta se destaca por gerar as seguintes métricas: *program spectrum*, número de *features* (funcionalidades) impactadas por método do projeto analisado e complexidade dos métodos de um software. *Program spectrum* se refere aos caminhos de execução de um sistema, destacando os pontos do *software* onde uma *feature* particular é executada [7].

Sendo assim, o *Feature-Trace* permite a análise da priorização e distribuição do esforço em testes de um software através do perfil operacional e outras métricas com a rastreabilidade oferecida pelo BDD. Além da ferramenta desenvolvida, enfatiza-se a metodologia

utilizada, que faz o uso da abordagem *BDD* para gerir informação relevante e até mesmo o perfil operacional de um *software*.

A Arquitetura do Feature-Trace pode ser descrita pelos seus três módulos distribuídos em uma estrutura cliente-servidor: *Automated Runtime Module* (ARM), Módulo de Análise e Módulo de Visualização. Na figura 2.1 podemos ter uma visão geral sobre essa arquitetura e cada módulo será descrito nas próximas seções.

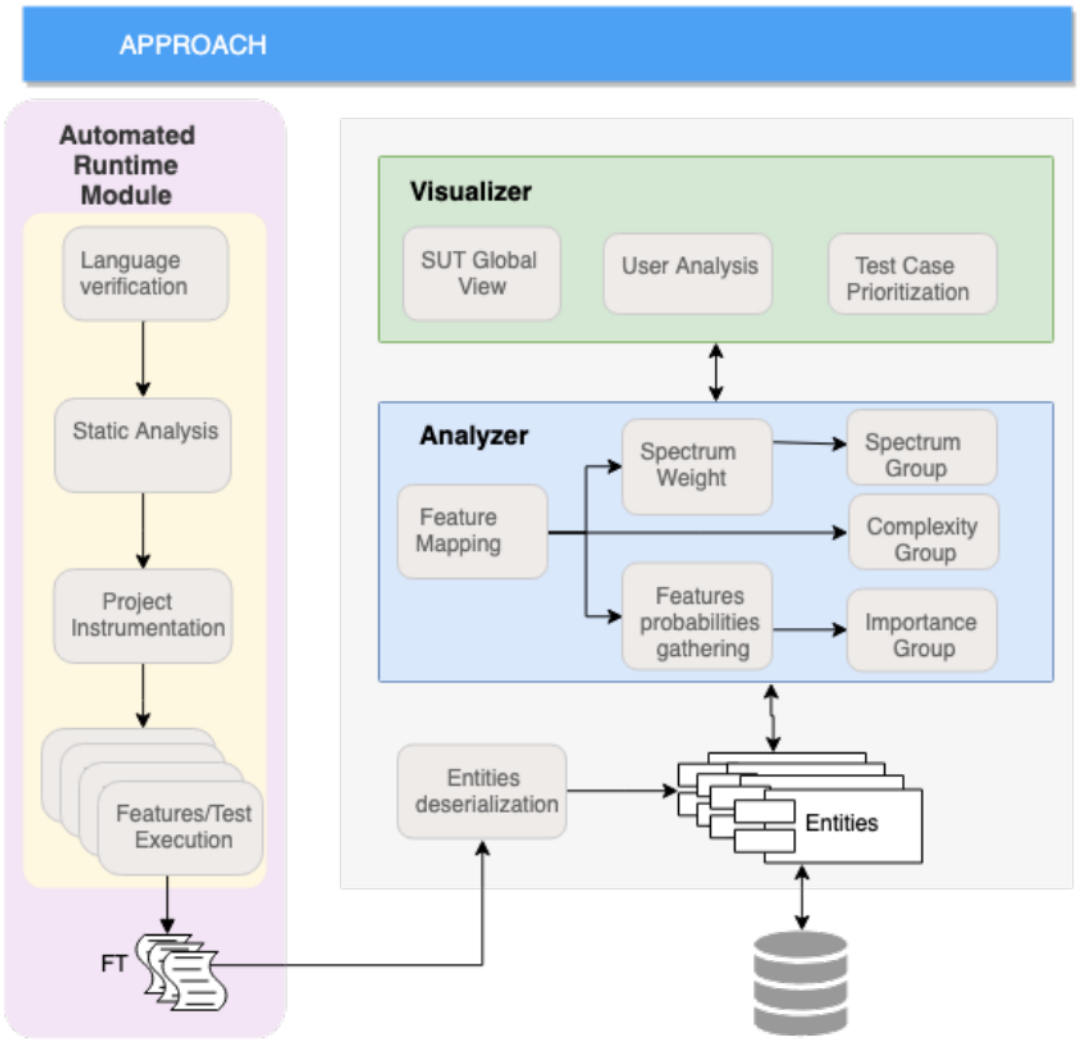


Figura 2.1: Arquitetura do Feature-Trace [1]

2.3.1 Automated Runtime Module

O ARM tem como objetivo principal a leitura estática e dinâmica do *software que está sendo testado*, extraindo dados e enviando essa informação para os próximos dois módulos,

de análise e visualização [1]. Entidades necessárias para formação do perfil operacional e outras métricas são recolhidas tanto dos testes BDD quanto do código fonte do SUT (sistema que está sendo testado). Esse módulo define o lado cliente da aplicação *Feature-Trace*.

Inicialmente, o ARM faz uma verificação da linguagem do *software* a ser testado, sendo que durante o tempo em que este trabalho está sendo feito o *Feature-Trace* pode analisar apenas projetos feitos em linguagem *Ruby*. O próximo passo compreende a extração de classes, métodos, testes e BDDs do projeto. Nesse processo de análise estática, a ferramenta também recolhe algumas métricas de complexidade, como Complexidade Ciclomática [8], *ABC Score* [9] e número de linhas de cada método. Por fim, a execução de testes e *features* mapeia o conjunto de métodos que são impactados e os envia para o módulo de análise.

2.3.2 Módulo de Análise

O módulo de análise é o primeiro que compõe o servidor do *Feature-Trace*, tendo como objetivo principal o mapeamento entre entidades do sistema que está sendo testado, relacionando os métodos deste sistema com os cenários e *features* dos BDDs [1]. Este módulo recebe a saída do ARM em forma de requisições de forma incremental e salva esses dados em um banco de dados que está presente no servidor, sendo que o mesmo é responsável pela extração de dados deste mesmo banco durante toda a execução da ferramenta.

Entre todas as análises de dados feitas por este módulo, destaca-se a elaboração do perfil operacional, que necessita da probabilidade de ocorrência de cada entidade como entrada específica para a conclusão do processo. De acordo com Musa [6], essa informação precisa ser obtida pela interação com o domínio em contexto, como por usuários experientes do *software*. Além do PO, um grande *output* desse módulo é o grafo que mostra a relação entre os métodos, cenários e *features* da aplicação testada.

2.3.3 Módulo de Visualização

O último componente do servidor do *Feature-Trace* é o módulo de visualização. Ele é responsável por apresentar os dados que são a saída do módulo de análise e uma visão geral do SUT [1]. Desta forma, também permite a análise de rastreabilidade do *software* a ser testado em forma de gráficos, destacando as informações geradas de forma mais visual . Dentre o conjunto de informações apresentadas pelo módulo, destaca-se o grupo de entidades com base em critérios como complexidade, *program spectrum* e o perfil operacional. Sendo assim o módulo de visualização serve como uma interface da

ferramenta, extremamente útil para a seleção e priorização de casos de testes de acordo com a estratégia desejada.

2.4 Containerização

Containerização é uma solução usada para facilitar a gestão e *deploy* (tornar um *software* disponível para o uso) de uma aplicação [10]. Um *container* é um pacote de *software*, um ambiente isolado que possui dependências que uma aplicação específica precisa para ser executada. É uma forma de virtualização que compartilha os recursos do *host* (sistema operacional), fazendo com que sejam mais leves e rápidas que outras formas de virtualização [11], já que o acesso a esses recursos pode ser limitado pelo SO.

De acordo com [12], *containers* são utilizados tanto para a elaboração de aplicações com arquitetura baseadas em componentes quanto em microsserviços. Isso pode ser explicado pelo fato de que *containers* trazem confiabilidade para a execução de aplicações, dado que o ambiente de execução é controlado e será o mesmo independente da máquina que o suporta.

Sendo assim, destaca-se os seguintes benefícios do uso de *containers* no desenvolvimento de software:

- Confiabilidade para execução de aplicações: ter um pacote de software que serve como ambiente controlado para uma aplicação é útil para execução em diferentes máquinas.
- Facilita a configurabilidade do *software*: o *setup* de um projeto que usa *containers* é facilitado, já que o *container* é configurado para uma aplicação específica.
- Contribui como documentação: os arquivos de configuração de *containers* para uma aplicação podem servir como uma documentação do que esta precisa para ser executada.

Containerização é um termo importante e necessário para assimilar as contribuições deste trabalho, sendo que a aplicação e benefícios deste conceito serão abordados durante a apresentação da implementação do mesmo.

2.5 DevOps

DevOps é o termo que organizações atribuem para a aplicação de um conjunto de técnicas de desenvolvimento ágil na área de operações a fim de otimizar o ritmo e qualidade de suas aplicações [13]. É um conjunto de esforços e práticas para aproximar times de

desenvolvimento e de operações. Porém, o termo é novo e carece de uma delimitação clara, sendo que várias pesquisas existentes propõem diversas definições diferentes para o *DevOps* [14].

O termo *DevOps* é comumente confundido apenas como práticas de automação de aplicações. Enquanto isso é realmente um dos benefícios e práticas aplicadas, o foco da aplicação do *DevOps* deve ser a construção de uma **cultura de colaboração** entre as equipes, de acordo com [15]. Para remover barreiras, silos e aproximar os times, essa cultura deve ser desenvolvida buscando entendimento total dos times sobre as aplicações de um mesmo ponto de vista geral, provendo transparência e confiança, evitando conflitos entre as áreas de desenvolvimento e operações [16].

2.5.1 Ativadores e resultados da adoção do DevOps

De acordo com [16], a adoção de *DevOps* envolve a relação entre as seguintes categorias específicas: cultura de colaboração, automação, agilidade, *continuous measurement*, *quality assurance*, resiliência, compartilhamento e transparência. Essas categorias são divididas entre "ativadores"e resultados, sendo que os ativadores são categorias que precisam ser aplicadas na adoção do *DevOps* e resultados são os efeitos e objetivos da adoção. Algumas categorias podem ser tanto ativadores quanto resultados, e cada uma será descrita nas próximas seções.

Cultura de Colaboração

Como citado anteriormente, é o ponto central e o objetivo da adoção do *DevOps*. Essa categoria foca em remover barreiras e silos entre os times de desenvolvimento e encoraja diretamente outros ativadores como a transparência e compartilhamento [16]. Também faz com que a equipe não se concentre apenas no desenvolvimento do *software*, mas também no produto de forma mais agregada e geral.

Automação

Procedimentos manuais são considerados candidatos para propagar formações de silos, dificultando a construção de uma cultura de colaboração [16]. Tarefas automatizadas tiram a responsabilidade de pessoas particulares, contribuindo diretamente com outras categorias como transparência e compartilhamento. Além disso, automações são ativadores que trazem confiabilidade para o desenvolvimento e evolução de um projeto, já que reduzem o retrabalho e por consequência o risco de falha humana. Dentre todos as práticas de automação, destacamos algumas a seguir que são recorrentemente relacionadas ao contexto de *DevOps* e se encaixam no contexto deste trabalho.

- **Continuous Integration**: Uma forma de automação bastante utilizada pela indústria atualmente é o *Continuous Integration* (CI), que é uma prática do desenvolvimento de *software* onde os membros de um time integram suas contribuições frequentemente [17]. Sendo assim, cada integração deve ser analisada por um *build* (processo de conversão de código fonte em *software* executável) automático da aplicação, incluindo execução de testes com o intuito de verificar se erros foram adicionados nas contribuições. Por esse motivo, CI costuma ser diretamente relacionado à execução de testes automatizados antes das integrações de novas contribuições em um projeto. Essa prática traz confiabilidade ao reduzir consideravelmente os erros de integração, fazendo com que o time possa focar nas contribuições de forma coesa e com ganho de agilidade.
- **Continuous deployment**: é uma automação e prática do desenvolvimento de *software* que está diretamente relacionada ao conceito popular de *DevOps*, já que durante a fase de *deploy* (ato de tornar um *software* disponível para uso) é onde os times de desenvolvimento e operação entram em contato. CD (Continuous deployment) pode ser definida pela automação do *deployment* de uma aplicação, geralmente quebrando contribuições em pequenas atualizações que serão incrementadas ao produto [18]. A entrega de *software* é uma clara manifestação de entrega de valor, onde transparece o impacto da automação de *deployment* no ganho de agilidade e confiabilidade no desenvolvimento de uma aplicação.
- **Microsserviços**: são constantemente citados como um aspecto do *DevOps* [19]. Mesmo não sendo uma forma direta de automação, uma arquitetura orientada a microsserviços prediz que os serviços devem ter *deployment* totalmente automatizado e independente, evidenciando que sua adoção está diretamente relacionada a automações. Nessa arquitetura, os chamados microsserviços possuem um objetivo bem definido, apresentando alta coesão e baixo acoplamento.

Agilidade

Quando pondera-se sobre a adoção do *DevOps*, agilidade é considerada como objetivo e benefício do processo. Com mais colaboração entre os times, técnicas de desenvolvimento ágil e toda a transformação de infraestrutura como código resultam naturalmente em um ganho de agilidade. Esse ganho fica claro quando cogita-se que técnicas como *continuous integration* e *continuous deployment* são comumente aplicados em um contexto de *DevOps*.

Continuous Measurement

De acordo com [16], *continuous measurement*, que é a tarefa de continuamente recolher métricas de uma aplicação e de sua infraestrutura, reforça a cultura de colaboração de uma equipe, dado que essa tarefa, antes uma responsabilidade do time de operações, transparece por todo a equipe que trabalha no projeto. Sendo uma categoria considerada tanto como um ativador quanto um resultado da adoção do *DevOps*, mostra-se necessária já que o ganho de agilidade resultante do processo acelera o crescimento de escopo e necessidade de recursos da aplicação.

Quality Assurance

Garantir a qualidade de uma aplicação sempre foi uma categoria que fomenta várias pesquisas e discussões no desenvolvimento de *software*. Como um ativador e objetivo, garantir a qualidade de um *software* aumenta a confiança da equipe no projeto, contribuindo com a construção da cultura de colaboração, de acordo com [16]. Com a agilidade resultante da adoção do *DevOps*, é imprescindível garantir que a qualidade da aplicação seja mantida, sendo que isso pode ser alcançado utilizando técnicas distintas como a análise estática do código fonte da aplicação e cobertura de testes.

Resiliência

Como um resultado esperado pela adoção do *DevOps*, de acordo com [16], resiliência representa a capacidade de uma aplicação para se adaptar a situações adversas. Conceitos e práticas que afetam esta resiliência podem ser exemplificados por automações como: recuperação automática, que é a capacidade de infraestrutura que recupera a aplicação após falhas; *auto scalling*, que é a distribuição dinâmica de recursos à aplicação de acordo com sua demanda.

Compartilhamento e Transparência

De acordo com [16], ambos são considerados ativadores na adoção do *DevOps* e estão diretamente relacionados como conceitos que focam em difundir informação entre os desenvolvedores de um produto. Com transparência e compartilhamento, todos os membros da equipe alcançam uma visão melhor do processo de desenvolvimento de produto, resultando em uma contribuição com a cultura de colaboração. Comitês de leitura, grupos de conversas e treinamentos são exemplos de ações que propagam essas categorias e resultam em passagem de conhecimento entre o time.

2.5.2 Metodologia para adoção do Devops

Adotar *DevOps* em um contexto já existente e complexo não é uma atividade trivial. Como aborda-se uma área que ainda demanda delimitações, não é claro como esse processo de adoção deve ocorrer. Sendo assim, [16] apresenta um modelo de adoção do *DevOps* que engloba todo o processo, desde a etapa de planejamento até a execução e resultado. Este modelo pode ser descrito em três etapas:

1. Primeiramente, a relevância da constituição de uma cultura de colaboração entre os membros da equipe e o objetivo da adoção do *DevOps* devem ser levantados pela organização.
2. Nessa etapa os ativadores relevantes para o contexto da equipe e do produto devem ser selecionados, almejando o objetivo da adoção e que contribuam com a cultura de colaboração de forma mais efetiva possível.
3. Por fim, verificar os resultados da adoção do *DevOps*, refletindo sobre o alinhamento com técnicas e conceitos comumente usados na indústria e comunidade de desenvolvimento de *software*, de acordo com a necessidade da organização.

Essa metodologia proposta por [16] e os princípios de DevOps levantados são imprescindíveis para a fundamentação das contribuições desse trabalho. Elabora-se nos próximos capítulos uma seleção destes princípios para a composição das propostas deste trabalho, de acordo com a pertinência de cada um destes para o contexto do Feature-Trace.

Capítulo 3

Fundamentação da Proposta

A configuração e utilização do Feature-Trace não é trivial. O usuário da ferramenta precisa configurar uma série de artefatos distribuídos tanto no cliente quanto no servidor:

- **Cliente**: Necessita-se que o usuário obtenha a aplicação cliente diretamente de um repositório no *GitHub* [20] (controlador de versões online) e a instalar localmente, lidando com várias versões de dependências que este *software* desenvolvido em linguagem de programação *Python* [21] necessita. Após a instalação, o usuário tem acesso a uma série de comandos disponíveis via linha de comando responsáveis pela análise do SUT.
- **Servidor**: a referida aplicação desenvolvida em *Python* foi construída como um projeto *web* com banco de dados, possuindo interface com usuário e utilizando o *Framework Django* [22] para sua elaboração. Para executar o servidor, o usuário precisa obter o código fonte de outro repositório no *GitHub* e iniciar a instalação localmente. Esse processo tem como dependência a configuração e criação de um banco de dados utilizando PostgreSQL [23], um sistema de gerenciamento para banco de dados relacionais. Após a configuração, o usuário inicia o projeto em um servidor local, para que o mesmo esteja pronto para receber requisições do cliente e funcionar como servidor da ferramenta.

Sendo assim, o usuário possui uma série de pré-requisitos e passos a seguir antes de utilizar o Feature-Trace. Toda essa burocracia torna a utilização da ferramenta complexa e retira o foco do *software* a ser testado e analisado. De acordo com [24], a usabilidade de um *software* de código aberto é geralmente uma das razões por limitações em sua distribuição. Portanto, essa complexidade na configurabilidade e usabilidade torna inviável o uso deliberado e em larga escala do Feature-Trace.

Além disso, o estado desacoplado e não encapsulado do Feature-Trace dificulta novas contribuições para o mesmo, algo prejudicial para uma ferramenta útil como esta. Entre-

gas de valor para o projeto não alcançam os usuários de forma natural, dado que o mesmo precisa obter código fonte da ferramenta diretamente em um ambiente local, restringindo a versão da ferramenta para cada usuário.

No decorrer deste capítulo, as soluções encontradas para a resolução destes problemas serão apresentadas, inicialmente passando pela proposta do trabalho sugerida e metodologia aplicada que fundamenta as contribuições.

3.1 Proposta

Para solucionar as limitações citadas do Feature-Trace, a proposta principal é a aplicação de uma série de conceitos de *DevOps* para tornar a ferramenta não apenas suscetível a contribuições, mas também aprimorando sua configurabilidade e usabilidade perante o usuário. Sendo assim, temos como objetivos principais:

- Tornar ágil e usual o processo de desenvolvimento de novas funcionalidades para o Feature-Trace.
- Possibilitar que novas contribuições ao Feature-Trace possam alcançar o usuário de forma natural e sem gargalos.
- Facilitar a configurabilidade e usabilidade da ferramenta, tornando-a utilizável no cotidiano de um desenvolvedor e diminuindo o grau de esforço para o uso do Feature-Trace.
- Alinhar o Feature-Trace a técnicas e conceitos utilizados pela indústria e comunidade de desenvolvimento de *software*.

Portanto, o conjunto de contribuições deste trabalho tem como objetivo aprimorar tanto a perspectiva de desenvolvimento quanto a do usuário final do Feature-Trace.

3.2 Metodologia Utilizada

A metodologia utilizada para a adoção de conceitos do DevOps no Feature-Trace consiste de três passos que guiam organizações para essa aplicação em um cenário industrial comum da área de desenvolvimento de software (metodologia proposta por [16]).

1. Levantou-se o objetivo da adoção do *DevOps*, sendo o mesmo já abordado na proposta deste trabalho na Seção 3.1. Além disso, mesmo com o contexto de equipe distinto do Feature-Trace, a constituição de uma cultura de colaboração tem sim uma relevância e importância, dado que colaboração é algo intrínseco ao desenvolvimento de um *software* de código aberto [25].

2. Elaborou-se um levantamento sobre os ativadores do *DevOps* apresentados na Seção 2.5.1, levando em conta o contexto da ferramenta e quais impactariam e contribuiriam com o objetivo desse trabalho. Os seguintes ativadores foram selecionados: automações, *quality assurance* e constituição de uma cultura de colaboração. Como resultados, tem-se como objetivo alcançar categorias como agilidade e resiliência.

3. Verifica-se os resultados da adoção do *DevOps* no Feature-Trace, ponderando sobre o alinhamento com a indústria e comunidade de desenvolvimento de *software*, como proposto por [16]. Esses resultados serão abordados de forma mais completa ao decorrer do trabalho.

Mesmo que o foco da adoção de *DevOps* seja a constituição de uma cultura de colaboração, algo dissimulado dado o contexto da ferramenta, a perspectiva de adoção dos demais ativadores como a automação é essencial para alcançar os objetivos deste trabalho.

Trata-se de contribuições em um projeto de software de código aberto, portanto alterações foram necessárias na aplicação da metodologia para adequar ao contexto do Feature-Trace, principalmente quanto à questão das equipes no qual é volátil em tal circunstância. Sendo assim, a adoção de princípios de *DevOps* no Feature-Trace através dos três passos citados anteriormente contam com alterações para a adequação apontada.

3.3 Ativadores e resultados esperados

Como descrito na segunda etapa da metodologia proposta, deve-se efetuar o levantamento de ativadores do *DevOps* pelo quais a adoção destes princípios seguirá. Com este intuito, a Tabela 3.1 apresenta um apanhado geral e sumarizado deste levantamento, correlacionando as contribuições selecionadas para cada ativador com os objetivos e resultados esperados para a adoção de *DevOps*.

Objetivos	Ativadores DevOps	Contribuições	Resultados Esperados
Tornar ágil e usual o processo de desenvolvimento de novas funcionalidades.	• Automação • Resiliência • Cultura de Colaboração	• Containerização do servidor do Feature-Trace • Aplicação de CI/CD no Servidor • Aplicação de CI no Cliente	Fluxo de desenvolvimento possui artefatos para o auxílio do desenvolvedor.
Possibilitar que novas contribuições ao Feature-Trace possam alcançar o usuário de forma natural.	• Automação	• Aplicação de CD no Servidor • Cliente do Feature-Trace como Pacote	Módulos com Deployment e hospedagem remoto, possibilitando lançamento de novas versões.
Facilitar a configurabilidade e usabilidade da ferramenta.	• Automação	• Aplicação de CD no Servidor • Cliente do Feature-Trace como Pacote	Novo fluxo de utilização da ferramenta, mais simples que anteriormente.
Alinhar o Feature-Trace a técnicas e conceitos utilizados pela indústria e comunidade de desenvolvimento de *software*.	• Automação • Resiliência • Cultura de Colaboração	• Containerização do servidor do Feature-Trace • Aplicação de CI/CD no Servidor • Aplicação de CI no Cliente • Cliente do Feature-Trace como Pacote	Feature-Trace agora utiliza técnicas e aplica conceitos amplamente utilizados.

Tabela 3.1: Apanhado geral da execução

Cada objetivo detalhado na tabela acima definiu ativadores que foram aplicados durante a adoção do *DevOps* através das contribuições descritas, culminando em resultados esperados correlacionados.

Aborda-se nas próximas seções a implementação das contribuições citadas na Tabela 3.1.

3.4 Containerização do servidor do Feature-Trace

Como descrito sobre a utilização do Feature-Trace nesse capítulo, o servidor apresenta várias dependências e configurações como pré requisitos para sua execução. Esse gargalo se torna ainda mais evidente para novos desenvolvedores que desejam contribuir com a ferramenta, dado que estes não conhecem o contexto do projeto, dificultando a curva de aprendizado sobre o Feature-Trace e prejudicando a agilidade de entrega de possíveis novas funcionalidades.

Visando uma solução para este problema, aplicou-se uma containerização no servidor do Feature-Trace utilizando a tecnologia *Docker* [26], uma plataforma que oferece diversos artefatos para o processo de aplicação e gerenciamento de *containers*.

Com essa solução, as dependências necessárias para a execução local do servidor limitam-se ao *Docker* e seus artefatos. Além disso, como citado na Seção 2.4, essa containerização contribui também com a documentação da aplicação, agregando com a agilidade do desenvolvimento do Feature-Trace.

Além do propósito citado , a aplicação de containerização para com o servidor da ferramenta demonstra alinhamento com práticas amplamente utilizadas. Portanto, a mesma é uma contribuição que impacta dois dos objetivos descritos na Tabela 3.1, sendo uma manifestação do ativador automação no contexto de *DevOps*. Os detalhes da implementação dessa containerização serão abordados nas seções seguintes.

3.4.1 Dockerfile: imagem do container

Nesse arquivo de configuração são descritas as instruções para a criação da imagem do *container*. Com os comandos declarados no *Dockerfile*, o *Docker* monta a imagem necessária para execução do *container*. Na Figura 3.1 a seguir esse arquivo de configuração é apresentado, declarando os pré-requisitos e configurações necessárias para que o *container* funcione. Essa configuração aponta para outro arquivo, o *entrypoint* que será abordado na próxima Seção.

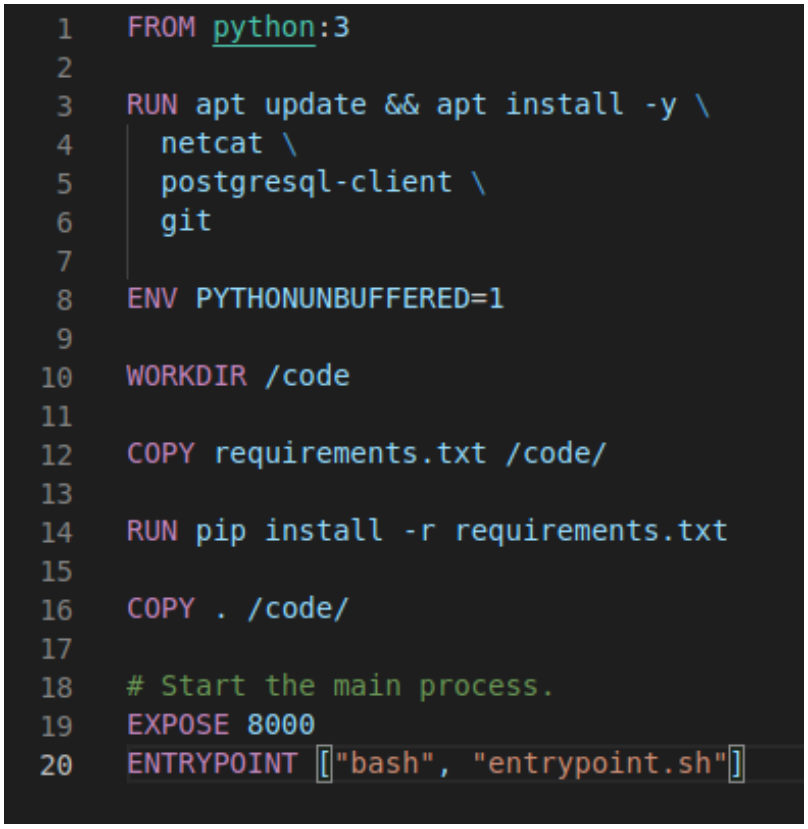
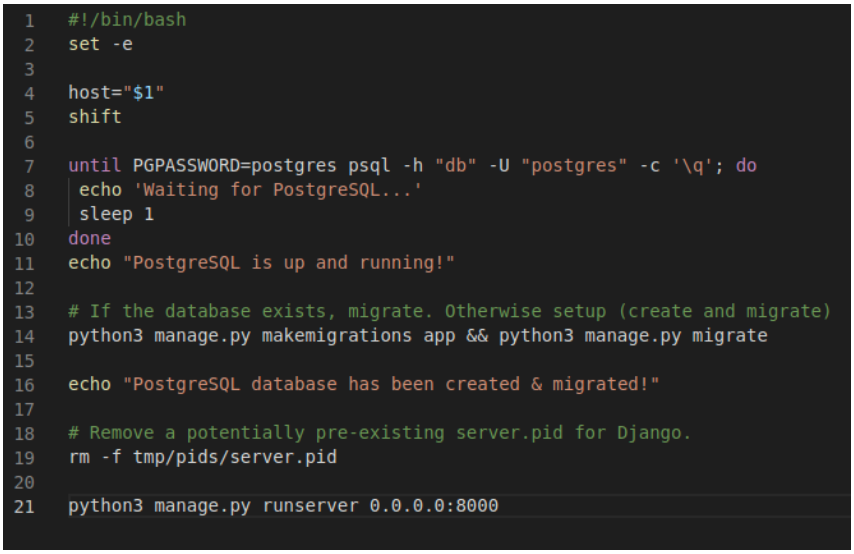


Figura 3.1: Dockerfile do servidor do Feature-Trace

3.4.2 Entrypoint

Nesta configuração uma série de comandos são declarados e podem ser analisados na Figura 3.2. Estas instruções descrevem definições que serão chamadas quando o *container* for executado. Neste arquivos temos tratamento de comunicação com o serviço do banco de dados (*PostgreSQL*) e o comando para iniciar o servidor *Django*.



```
#!/bin/bash
set -e

host="$1"
shift

until PGPASSWORD=postgres psql -h "db" -U "postgres" -c '\q'; do
 echo 'Waiting for PostgreSQL...'
 sleep 1
done
echo "PostgreSQL is up and running!"

# If the database exists, migrate. Otherwise setup (create and migrate)
python3 manage.py makemigrations app && python3 manage.py migrate

echo "PostgreSQL database has been created & migrated!"

# Remove a potentially pre-existing server.pid for Django.
rm -f tmp/pids/server.pid

python3 manage.py runserver 0.0.0.0:8000
```

Figura 3.2: Entrypoint do servidor do Feature-Trace

Ressalta-se que a existência do arquivo de *Entrypoint* encapsula as definições iniciais para a execução do *container*, exemplo claro de contribuição para com a documentação do Feature-Trace.

3.4.3 Docker-Compose

Por fim, temos o arquivo *docker-compose*, arquivo que define a configuração do *Docker Compose* [27], uma ferramenta para definição e execução de múltiplos *Docker containers*. Na configuração descrita na figura 3.3, cada *container* é definido como serviço, sendo que com apenas alguns comandos pode-se criar e executar todos esses serviços ao mesmo tempo, trazendo simplicidade para iniciar o servidor localmente.

```yaml
version: "3.3"

services:
  db:
    image: postgres
    volumes:
      - postgresdata:/var/lib/postgresql/data
    environment:
      - POSTGRES_DB=postgres
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=postgres
  web:
    build: .
    command: python manage.py runserver 0.0.0.0:8000
    volumes:
      - .:/code
    ports:
      - "8000:8000"
    depends_on:
      - db
volumes:
  postgresdata:
    driver: local
```

Figura 3.3: Docker Compose do servidor do Feature-Trace

Neste caso, os serviços declarados pelo *Docker Compose* são o banco de dados e a própria aplicação em si, estes que em conjunto definem o servidor do Feature-Trace. Ao executar estes serviços o servidor é iniciado localmente, completando a lógica de implementação da containerização do mesmo e apresentando uma simples configuração para este módulo do Feature-Trace.

3.5 Aplicação de CI/CD no Servidor

Continuous Integration e *Continous Deployment* são regularmente lembrados quando aborda-se sobre automações, como citado na Seção 2.5.1. O uso de CI/CD é uma prática amplamente utilizada na indústria do desenvolvimento de *software* e sua aplicação no contexto do Feature-Trace mostra-se interessante, dado os objetivos desse trabalho e a adoção do *DevOps*. Nas próximas seções, são descritas as motivações e implementações dessas práticas.

3.5.1 Continuous Integration

Um dos objetivos deste trabalho é tornar o Feature-Trace mais suscetível a contribuições, como descrito na tabela 3.1. Para que o servidor tenha um fluxo ideal de desenvolvimento de novas funcionalidades, foi proposta uma configuração de *Continuous Integration* para a

ferramenta, que executa um *build* da aplicação em conjunto com a execução de testes. Essa configuração traz confiabilidade e agilidade para o desenvolvimento de novas contribuições.

O CI foi elaborado com o auxílio da funcionalidade *GitHub Actions* [28] oferecida pelo *GitHub*, controlador de versões online onde o repositório do servidor do Feature-Trace encontra-se. Através do *GitHub Actions*, declara-se arquivos de configurações que definem ações para automações de fluxos de trabalho, comumente utilizadas para definição de CI e CD.

Na figura 3.4 visualiza-se a configuração do CI, onde definiu-se uma ação que será executada em *pull requests*, um método frequentemente utilizado para o envio de contribuições de um *software*. Nesta ação um ambiente é preparado para a execução da ferramenta (build automático) seguido da execução de testes do servidor.

```yaml
name: Test Pull Request

on: pull_request

jobs:
  test:
    runs-on: ubuntu-latest
    services:
      postgres:
        image: postgres:11@sha256:85d79cba2d4942dad7c99f84ec389a5b9cc84fb07a3dcd3aff0fb06948cdc03b
        ports: ['5432:5432']
        options: >-
          --health-cmd pg_isready
          --health-interval 10s
          --health-timeout 5s
          --health-retries 5
    steps:
    - uses: actions/checkout@v2
    - name: Set up Python 3.x
      uses: actions/setup-python@v2
      with:
        python-version: '3.x'
        architecture: 'x64'
    - name: psycopg2 prerequisites
      run: sudo apt-get install libpq-dev
    - name: Install dependencies
      run: |
        python -m pip install --upgrade pip
        pip install -r requirements.txt
    - name: Run migrations
      run: python manage.py makemigrations app && python manage.py migrate
    - name: Run tests
      run: pytest tests/
```

Figura 3.4: Arquivo de configuração do CI do servidor

Logo, o *Continuous Integration* sempre será executado quando um contribuidor do Feature-Trace criar um *pull request*, sendo que para a integração dessa contribuição através dessa requisição é necessária a aprovação do CI, executando o *build* automático e os testes sem erros. Isso garante que a nova contribuição não afete negativamente funcionalidades já existentes no projeto.

Além de ser uma clara manifestação de uma aplicação do ativador automação do *DevOps* como descrito em 2.5.1, o CI também aborda o *Quality Assurance* para com o Feature-Trace, dado que neste processo verificamos e executamos testes automatizados.

3.5.2 Continuous Deployment

Como citado na Seção 2.5.1, entrega de *software* significa entrega de valor. Atualmente, o servidor do Feature-Trace não possui *deployment*, pois não existe servidor online do mesmo. Isso impacta diretamente o usuário da ferramenta e as novas contribuições no servidor, objetivos levantados na tabela 3.1. Como solução para esta limitação, aplicou-se uma configuração de *Continuous Deployment*, mais uma manifestação de automação remetente ao *DevOps*, novamente utilizando o *GitHub Actions* em conjunto a hospedagem do servidor de forma remota.

Portanto, o servidor passa a ser alocado com hospedagem *online*, removendo a necessidade de instalação e configuração deste lado da aplicação na utilização usual do Feature-Trace. Além disso, com o CD, novas contribuições podem alcançar os usuários do Feature-Trace de forma mais natural e com maior agilidade.

A seguir, na Figura 3.5, observa-se a configuração da ação do *GitHub Actions* que define o CD do servidor do Feature-Trace.

```yaml
on:
  push:
    branches: [ master ]

name: Deploy to Heroku Production

jobs:
  deploy:
    name: Deploy
    runs-on: ubuntu-latest
    steps:
    - name: Checkout
      uses: actions/checkout@v2
    - name: Heroku deploy
      uses: akhileshns/heroku-deploy@v3.12.12
      with:
        heroku_api_key: ${{ secrets.HEROKU_API_KEY }}
        heroku_app_name: ${{ secrets.HEROKU_PRODUCTION_NAME }}
        heroku_email: ${{ secrets.HEROKU_EMAIL }}
```

Figura 3.5: Arquivo de configuração do CD do servidor

No envio de contribuições na ramificação principal do repositório, o *deployment* automático no provedor de hospedagem online *Heroku* [29] é efetuado. Além disso, essa hospedagem ocorre em dois servidores distintos: o chamado servidor *staging* (para teste e homologação) e outro de *produção* (servidor principal que é disponibilizado para usuários finais). Como [30] expressa, essa distinção entre servidores é comum para a indústria de desenvolvimento de *software*, dado que a utilização do servidor *staging* é útil para testes e verificação de novas funcionalidades.

Uma configuração similar a da Figura 3.5 foi elaborada para o servidor *staging*, diferindo em variáveis e configurações específicas, sendo que esta ação é executada ao envio de contribuições em uma ramificação secundária do repositório no *GitHub*.

Em conjunto, o *Continuous Integration* e *Continuous Deployment* trazem automação ao processo de entrega de valor ao usuário final, bem como agilidade no desenvolvimento de novas funcionalidades e um alinhamento com práticas comumente utilizadas pela indústria e comunidade de desenvolvimento de software, contribuindo diretamente nos objetivos da adoção de *DevOps* abordados na tabela 3.1.

3.6 Refatorações no Cliente

Dada as alterações feitas no servidor, precisa-se que o módulo cliente do Feature-Trace permaneça em sincronia com o mesmo, funcionando perfeitamente em prol da ferramenta. Com este propósito, algumas refatorações descritas a seguir foram necessárias.

3.6.1 Trabalhando com servidor remoto

Como citado na Seção 3.5.2, uma das contribuições deste trabalho é o *deployment* do servidor, fazendo com que ele funcione com hospedagem *online*, sendo que o mesmo anteriormente deveria ser usado na máquina do usuário de forma local. Dada essa nova abordagem do servidor, o cliente necessita de refatorações para operar e enviar suas requisições de forma dinâmica para servidores distintos de acordo com a necessidade e perfil do usuário, seja este usuário final ou desenvolvedor da ferramenta.

Sendo assim, o cliente passou por uma refatoração para que este funcione com uma URL (endereço virtual de uma página) de servidor variável, recebida como parâmetro na execução de métodos do Feature-Trace via linha de comando. Na Figura 3.6 pode-se analisar a adição da opção de URL como parâmetro, mantendo como padrão o valor do servidor local, escolha feita para manter o Feature-Trace retroativo com seu funcionamento anterior. Essa configuração pertence a uma biblioteca do *Python* chamada *Click* [31], que aborda e define métodos de linha de comando úteis para *softwares* com esse tipo de comunicação com o usuário.

```
58  @click.option(
59      '--url',
60      '-u',
61      default='http://localhost:8000',
62      help='This option specify the target server url'
63  )
```

Figura 3.6: URL do servidor como parâmetro no Cliente

Todas os métodos do cliente que abordavam requisições com o servidor passaram por refatorações para receberem esse novo parâmetro, além dos métodos que os chamam em suas execuções, de forma cascateada. Ainda, outra nova funcionalidade que envolve a URL variável do servidor foi necessária para estes métodos e será descrita a seguir. Será descrita a seguir outra nova funcionalidade fundamental para estes métodos que envolve a URL variável do servidor.

3.6.2 Backoff Exponencial

Durante os testes e análise da comunicação entre o cliente e o servidor com nova abordagem, verificou-se um problema em potencial: o cliente até então não possui algum tipo de tratamento em caso de erros de comunicação com o servidor, algo imprescindível para *softwares* com esse tipo de funcionamento, remetente a microsserviços. Como o cliente do Feature-Trace envia requisições de forma incremental para a execução de vários de seus métodos, um erro de qualquer tipo nessa comunicação faz com que a execução da análise de um *software* seja interrompida. Esse fato motivou a implementação de um *Backoff Exponencial* [32], técnica consolidada e frequentemente utilizada para ditar intervalos entre novas tentativas de comunicação após erros de requisições.

Na Figura 3.7, a lógica de *Backoff Exponencial* adotada para esta solução é demonstrada através de um fluxograma.

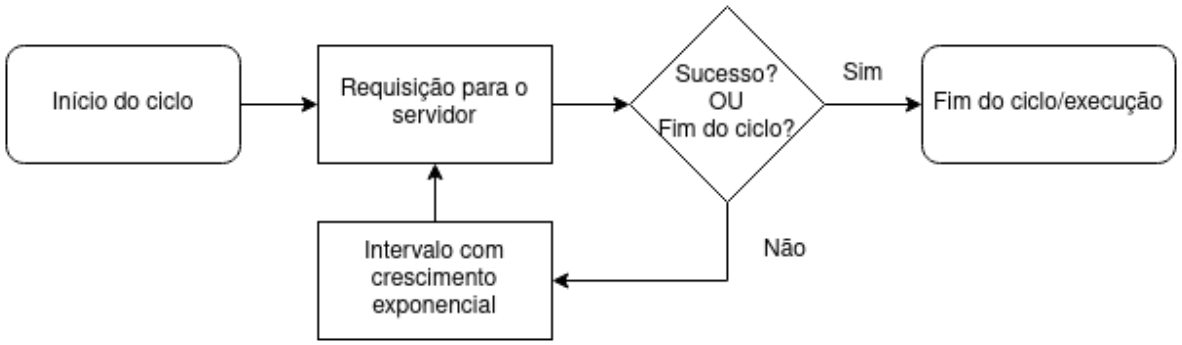


Figura 3.7: Lógica de Backoff Exponencial adotada

Em resumo, um processamento é efetuado em um *loop* (ciclo), sendo que esse ciclo é encerrado naturalmente ou após o sucesso do processamento. Em caso de falha, um intervalo é chamado antes que o processamento receba uma nova tentativa. É comum para implementações desse algoritmo que o intervalo entre novas tentativas do procedimento sejam cada vez maiores, em crescimento exponencial, comportamento que explica o nome do algoritmo em questão. O fim de ciclo de forma natural, dando um limite de tentativas para o algoritmo, geralmente é definido após um tempo limite de intervalo de tentativas ou por definição de número de interações do ciclo, sendo que esta última definição foi abordada para este trabalho.

A seguir, a figura 3.8 detalha a implementação de código do *Backoff Exponencial* no cliente do Feature-Trace, adotado para os diversos métodos que fazem requisições para o servidor .

```
429 for retry in range(1, 4):
430     try:
431         request = requests.post(url + "/covrel/update_spectrum",
432                                 json=json_string)
433         print(request.status_code, request.reason)
434         return request.status_code
435     except:
436         print("Connection refused by the server... Waiting to try again")
437         time.sleep(3**retry + random.uniform(0, 1))
438         print("Trying again for the " + str(retry) + "° time")
439     break
440 else:
441     print("Could not connect to server...exiting")
```

Figura 3.8: Backoff Exponencial no Feature-Trace

Em um ciclo de três execuções, o código acima descreve uma funcionalidade de *try* (tente) e *except* (exceto) para que o método siga com a execução da requisição para com o servidor, mas capturando exceções (erros de execução). Instruções são definidas para possíveis erros, descrevendo o comportamento de atraso através de intervalos antes de uma nova tentativa de requisição, além de mensagens ao usuário sobre este procedimento. Essas instruções também ditam o crescimento exponencial do intervalo a cada nova tentativa.

Além de solucionar a quebra de execução do Feature-Trace causada por erros de comunicação com o servidor, esta solução apresenta abordagem necessária para esse tipo de comunicação. Isso ocorre visto que diversos tipos de erros de requisição não controlados por desenvolvedores do Feature-Trace podem ocorrer principalmente no uso do servidor hospedado remotamente, nova abordagem proposta por este trabalho.

Na Figura 3.9 a seguir pode-se ver o algoritmo de *Backoff Exponencial* em ação, referente a caso ocorrido durante testes com o servidor remoto na execução de um método do Feature-Trace que captura métodos de um SUT.

```
Get Method:     def self.all_status

Number of executed Methods:  453
Connection refused by the server... Waiting to try again
Trying again for the 1° time
200 OK
```

Figura 3.9: Exemplo de ocorrência do Backoff Exponencial

3.7 Aplicação de CI no Cliente

Além de refatorações pertinentes ao cliente, também é objetivo deste trabalho que a adoção de *DevOps* esteja presente no cliente do Feature-Trace. Seguindo a metologia e os ativadores selecionados para essa adoção, analisou-se que a adição de uma configuração de *Continuous Integration* seria apropriado para o cliente, dados motivos similares da mesma abordagem no servidor, descritos na Seção 3.5.1.

A necessidade de uma automação como o CI no cliente transparece na análise de um dos objetivos descritos na tabela 3.1: deixar o Feature-Trace suscetível a contribuições e com fluxo de desenvolvimento ideal para novas funcionalidades. Para contribuir com este objetivo no ponto de vista do cliente, essa configuração surge como solução. A Figura 3.10 a seguir demonstra a implementação deste CI.

```yaml
name: Test Pull Request

on: pull_request

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v2
    - name: Set up Python 3.x
      uses: actions/setup-python@v2
      with:
        python-version: '3.x'
        architecture: 'x64'
    - name: Install dependencies
      run: |
        python -m pip install --upgrade pip
        pip install -r requirements.txt
        pip install .
    - name: Run tests
      run: pytest trace_feature/tests/ -v --cov
```

Figura 3.10: Arquivo de configuração do CI do cliente

Novamente, foi utilizada a ferramenta *GitHub Actions* [28] que permitiu definir uma ação que será executada em *pull requests*, configuração similar a da desenvolvida para o CI do servidor. Foram desenvolvidas configurações para o *build* automático específicas para o tipo de aplicação do cliente em conjunto à execução de testes presentes. Isso traz objetivos deste trabalho, como confiabilidade e agilidade, para o cliente do Feature-Trace.

3.8 Cliente do Feature-Trace como Pacote

Seguindo a implementação do CI no cliente, analisou-se como a entrega de valor ao usuário é feita nesse módulo do Feature-Trace. Como citado anteriormente neste capítulo, o cliente do Feature-Trace encontra-se em estado desacoplado e não encapsulado, sendo que o mesmo necessita que seu código fonte seja baixado e executado diretamente na máquina do usuário, em uma configuração completamente manual e não ideal.

Dada a natureza do cliente que define métodos para o usuário via linha de comando para análise de um *software*, a seguinte solução mostrou-se pertinente para estes problemas: a publicação do cliente do Feature-Trace como pacote *Python*. Neste caso, um pacote é uma biblioteca que pode ser hospedada remotamente e baixada na máquina de um usuário sem a necessidade que o usuário faça processos manuais já citados.

Essa solução removeria a necessidade de um grande passo e gargalo no fluxo de utilização da ferramenta, que é a obtenção do código fonte do cliente via repositório. Adicionalmente, essa proposta traz a questão de *deployment* para o módulo cliente, remetendo a uma arquitetura de microsserviços para com o servidor e por fim alinhando a ferramenta com os objetivos e conceitos da adoção do *DevOps*.

Além de contribuir com a configurabilidade e usabilidade da ferramenta, esta solução também contribui com a entrega de *software* e desenvolvimento de novas funcionalidades para o cliente, objetivos abordados na tabela 3.1. Com o cliente em forma de pacote, os desenvolvedores do Feature-Trace podem elaborar publicações de novas versões para o mesmo, entregando valor aos usuários de forma ideal.

Portanto, publicou-se o cliente do Feature-Trace como pacote no *PyPi* [33], repositório de *softwares* para *Python* amplamente utilizado pela comunidade com qual pode-se pesquisar, instalar e publicar pacotes. O *PyPi* exige algumas configurações para que um *software* possa ser lançado como pacote em seu repositório. A seguir, na Figura 3.11, analisa-se a declaração de configuração principal requisitada para essa publicação.

```
from setuptools import setup

with open("README.md", "r") as fh:
    long_description = fh.read()

setup(
      name='trace_feature',
      packages=['trace_feature', 'trace_feature.core', 'trace_feature.core.ruby', 'trace_feature.core.features'],
      version='1.1',
      description='A lib to trace bdd features.',
      long_description=long_description,
      long_description_content_type="text/markdown",
      url='https://github.com/vitorbribas/trace_feature',
      download_url = 'https://github.com/vitorbribas/trace_feature/archive/refs/tags/v_1.1.tar.gz',
      author='Rafael Fazzolino',
      author_email='fazzolino29@gmail.com',
      license='MIT',
      keywords = ['BDD', 'Trace'],
      py_modules=['trace_feature'],
      install_requires=[
            'Click==7.0',
            'gherkin-official==4.1.3',
            'requests==2.21.0',
            'ez-setup==0.9'
      ],
      entry_points='''
            [console_scripts]
            trace-feature=trace_feature.trace_feature:trace
      '''
)
```

Figura 3.11: Principal Arquivo de configuração do Pacote

Neste arquivo declaram-se informações necessárias para uma publicação, como a versão atual do pacote, as dependências necessárias para a execução do mesmo, sua descrição, o endereço do repositório, licença, informações do autor do *software* e demais declarações. Com essas informações, o *PyPi* permite a publicação de uma versão do pacote, assim como a atualização de um existente. Em conjunto com algumas refatorações e declarações ao longo do projeto, publicou-se o cliente como pacote e este está disponível como *Trace Feature* [34] no *PyPi*.

Dessa forma, efetuou-se a publicação como pacote do cliente do Feature-Trace, tornando-o encapsulado e acoplado, suscetível a contribuições. Como este sendo o último passo da adoção do *DevOps*, fechamos o ciclo de aplicação de ativadores escolhidos na metodologia da Seção 2.5.2 e também o estágio de implementação deste trabalho.

Capítulo 4

Resultados Obtidos

Nas próximas seções serão descritos os resultados obtidos com a adoção de conceitos de *DevOps* para com o Feature-Trace, além do detalhamento do estudo de caso efetuado para validação das contribuições elaboradas que impactaram a perspectiva de usuário.

4.1 Resultados da adoção de princípios DevOps

Como descrito pela metodologia utilizada para a adoção do *DevOps* no Feature-Trace, apresentada na seção 3.2, verificam-se os resultados desta adoção como último passo. Tem-se como objetivo atingir categorias como agilidade e resiliência através da aplicação de ativadores do *DevOps*, além do alinhamento com práticas da indústria e comunidade de desenvolvimento de *software*.

Dada a aplicação de automações como *Continuous Integration* e *Continuous Deployment*, containerização, *deployment* dos artefatos do Feature-Trace (cliente e servidor) em forma encapsulada e as demais contribuições deste trabalho, afirmar-se que o alinhamento com técnicas e práticas amplamente utilizadas foi atingido, além de aumento na agilidade de desenvolvimento da ferramenta. Isso transparece quando se analisa o novo fluxo de desenvolvimento para a ferramenta, que agora oferece artefatos importantes para o auxílio deste processo.

Concluiu-se também que um certo grau de resiliência para com a ferramenta foi atingido, objetivo da aplicação de ativadores do *DevOps*. Com a abordagem de *deployment* no Feature-Trace, seus módulos possuem resiliência garantida pela hospedagem remota. Isso se aplica principalmente ao servidor, dado sua natureza mantenedora de dados.

A agilidade e resiliência resultados da adoção do *DevOps* estão correlacionadas e manifestam os resultados esperados da tabela 3.1. Além disso, cada contribuição adotada de acordo com a metodologia proposta e conceitos do *DevOps* impacta os objetivos e resultados abordados nesta tabela.

Por fim, levanta-se uma reflexão sobre a construção de uma cultura de colaboração para o Feature-Trace. Avalia-se este ponto mediante os resultados que abordam o contexto de desenvolvimento da ferramenta, sobre o fluxo de elaboração de contribuições para a mesma e uso de técnicas modernas amplamente utilizadas, descritos na tabela 3.1. Estes manifestam um esforço para com a elaboração da cultura de colaboração, visando que o Feature-Trace continue evoluindo e receba contribuições, dado sua natureza de *software* de código aberto.

Com isso, encerra-se a aplicação da metodologia selecionada para a adoção do *DevOps* no Feature-Trace. Como descrito na Seção 3.1, este trabalho também tem como objetivo impactar a configurabilidade e usabilidade da ferramenta, atingindo a perspectiva do usuário. Portando, aborda-se a seguir as mudanças e contribuições nesse escopo.

4.1.1 Novo fluxo de utilização

Devido às contribuições e alterações promovidas ao cliente e servidor do Feature-Trace, elaborou-se um novo fluxo de utilização da ferramenta para o usuário final. Em contrapartida ao fluxo inicial da ferramenta descrito na seção 3, tem-se a seguinte descrição para o uso e tratamento de cada um dos módulos do Feature-Trace:

- Cliente: Segere-se a criação de um ambiente isolado e controlado para a integração da ferramenta e em seguida a instalação do cliente publicado como pacote, através de um comando que lida também com as dependências necessárias para esta aplicação.
- Servidor: Em posse da URL do servidor remoto, o usuário é dispensado da necessidade de instalação ou configuração do servidor.

A remoção da instalação do servidor e do acesso ao código fonte do cliente retira gargalos e burocracias para a utilização da ferramenta. Por fim, após a execução de aproximadamente três comandos, o usuário tem acesso aos métodos do Feature-Trace via linha de comando. Esta configuração pode ser analisada a seguir.

```
virtualenv -p python3 env
source env/bin/activate
pip install trace-feature
```

Sendo que o primeiro comando cria o ambiente virtual isolado e controlado para a integração da ferramenta, o segundo acessa esse ambiente e por último tem-se o comando que instala o pacote do Feature-Trace. A instalação de pré-requisitos como os artefatos do *Python* são de responsabilidade do usuário e precedem os comandos demonstrados

acima. Mais detalhes do procedimento de instalação serão verificados na descrição do pacote disponível no domínio do *PyPi* [34].

Detalha-se a seguir o estudo de caso elaborado para validação deste novo fluxo de utilização do Feature-Trace.

4.2 Estudo de Caso

Para a validação do novo fluxo de utilização da ferramenta e demais contribuições para com a perspectiva do usuário descritas na tabela 3.1, efetuou-se um estudo de caso em forma de sessão com desenvolvedores de *software*. Nessa avaliação, os desenvolvedores utilizaram o Feature-Trace para analisar projetos em que atuam como contribuidores.

O principal objetivo deste estudo é avaliar a configurabilidade e usabilidade da ferramenta na perspectiva do usuário final, refletindo-se sobre a contribuição deste trabalho neste escopo.

Descreve-se neste capítulo a elaboração desse estudo de caso, detalhando o planejamento desta avaliação, sua execução e análise de resultados obtidos.

4.2.1 Planejamento da avaliação

Para o planejamento e organização da avaliação com desenvolvedores, foram elaboradas quatro etapas, sendo que cada uma será descrita nas seções seguintes.

Apresentação do Feature-Trace

Nessa primeira etapa da sessão prática apresenta-se de maneira detalhada o Feature-Trace e seu propósito. Essa apresentação contém, de forma geral:

- Fundamentação e explicação sucinta do Feature-Trace, abordando a arquitetura da ferramenta.
- Apresentação do propósito e vantagens da utilização da ferramenta, em conjunto com seus resultados e métricas disponibilizadas.
- Explicação das motivações e contribuições deste trabalho.

Com essa introdução e fundamentação do Feature-Trace, almeja-se a contextualização da ferramenta, pré-requisito para o guia de utilização inicial da mesma, descrita na Seção seguinte.

Guia simplificado para uso do Feature-Trace

Etapa que tem como objetivo a demonstração da utilização do Feature-Trace, abordando o fluxo de utilização e resultados oferecidos pela ferramenta. Porém, a instalação e configuração da mesma não será apresentada, medida que tem como motivação o não enviesamento dos desenvolvedores para pontos de avaliação deste estudo de caso.

Prática: utilização da Ferramenta

Este procedimento é o que possui maior duração, onde tem-se a execução da parte prática da sessão envolvendo o teste do Feature-Trace. Envia-se os endereços eletrônicos do pacote *Python* e servidor remoto do Feature-Trace para os participantes, explicando que devem seguir as instruções de instalação e configuração disponíveis nos mesmos. Dúvidas recorrentes e pontuais levantadas pelos desenvolvedores serão respondidas em auxílio e acompanhamento dos mesmos.

Envio de Questionário e encerramento da sessão

Por fim, um questionário é enviado para os desenvolvedores que utilizaram a ferramenta com o objetivo de obter dados e comentários acerca da utilização do Feature-Trace.

Utiliza-se na maioria das perguntas a escala *Likert*, regularmente utilizada para obtenção de respostas escaláveis nesses tipos de questionários. Estas podem ter como resposta números de um a cinco, sendo que cada valor está associado a um significado compreensível para o avaliado. Essas questões são descritas no Anexo I deste trabalho e serão abordadas detalhadamente nas próximas seções.

A seguir, descreve-se a execução desse planejamento de sessão de avaliação com desenvolvedores.

4.2.2 Execução

A execução do estudo de caso em forma de sessão prática seguiu o planejamento descrito, sendo elaborada remotamente via vídeo chamada.

Os participantes deste estudo de caso foram alunos da disciplina de Engenharia de Software, oferecida pelo Departamento de Ciência da Computação da Universidade de Brasília. Para esta disciplina, os alunos formaram grupos de contribuidores que necessitavam desenvolver funcionalidades em um projeto mediante a metodologia BDD. Cada grupo era responsabilizado por uma *feature* BDD que seria analisada pelo Feature-Trace na execução da sessão prática.

No total, efetuou-se a sessão prática em aproximadamente 1 hora e 30 minutos. Destaca-se que, como esperado, a etapa prática que aborda a utilização da ferramenta

teve a maior duração dentre todas, próximo de 1 hora. Este fato é explicado pelo fato de que esta etapa apresenta interação com os usuários.

As dúvidas apresentadas pelos desenvolvedores concentraram-se sobre o funcionamento do Feature-Trace, sua usabilidade e comandos que analisam o *software* a ser testado. Além disso, levantou-se algumas dúvidas sobre conceitos de fundamentação da ferramenta, como o perfil operacional e arquitetura da ferramenta.

Algumas dificuldades manifestaram-se através de erros de execução da ferramenta, sendo que estes serão abordados nas próximas seções.

Por fim, o questionário descrito no Anexo I foi enviado aos desenvolvedores que participaram da sessão prática, sendo que a análise dos resultados obtidos é abordado a seguir.

4.2.3 Resultados obtidos

Os resultados obtidos para o estudo de caso expressam-se através de onze respostas do formulário enviado após o encerramento da sessão prática. Para a organização da análise destas respostas, dividiu-se as questões em categorias distintas, cada uma com um objetivo específico. Detalha-se a seguir essas categorias em conjunto com a análise de respostas de cada pergunta, seguido de um apanhado geral dos resultados obtidos mediante o estudo de caso.

Levantamento de perfil do participante

O objetivo desse grupo de questões era o levantamento do perfil de cada participante. Esta informação auxilia o entendimento e análise de respostas providas pelas demais questões. A seguir, descreve-se a pergunta que encaixa-se nesta categoria.

P4 - Quando você pensa em utilizar uma ferramenta, avalia o quão trabalhoso será a integração da mesma em seu ambiente.

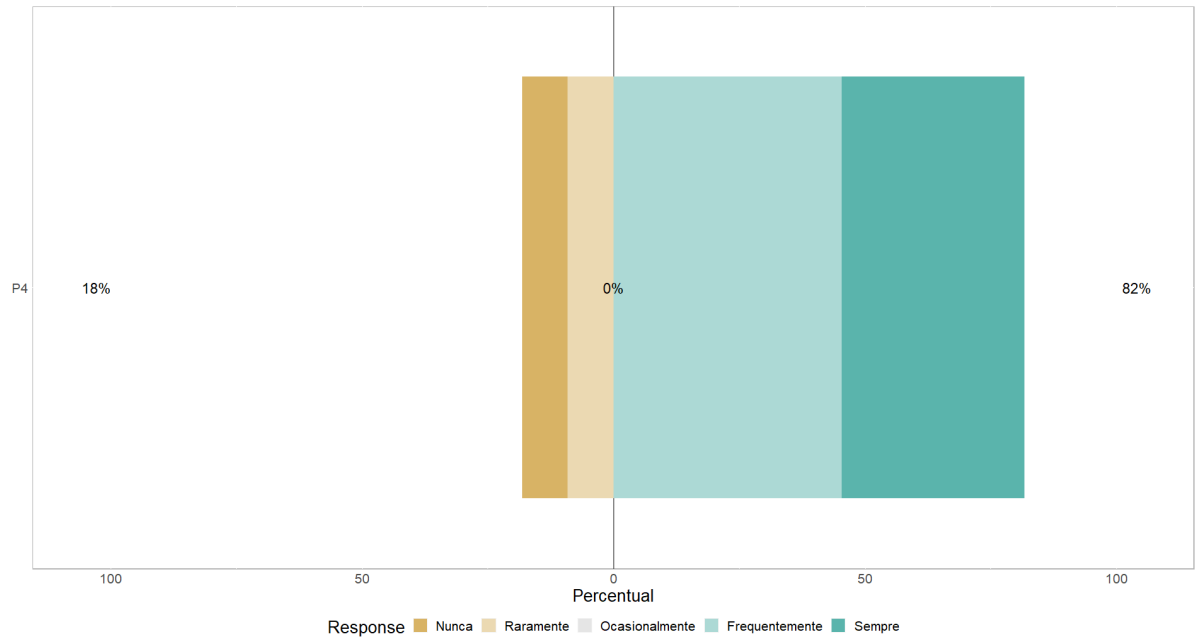


Figura 4.1: Pergunta sobre perfil do participante

Almejou-se com essa pergunta verificar o quão importante o custo de uso de uma ferramenta é para o participante. Verifica-se na figura 4.1 que a maioria destes consideram sim o esforço para a utilização de uma determinada ferramenta. Com isso, concluiu-se que os perfis dos participantes da sessão prática são ideais para esta pesquisa, pois estes em sua maioria preocupam-se com a configurabilidade, usabilidade e integração de uma ferramenta.

Validação de contribuição deste trabalho

As questões que compõem esta categoria aspiram a validação de um objetivo específico deste trabalho: o aprimoramento de configurabilidade e usabilidade da ferramenta, diminuindo o grau de esforço para o uso do Feature-Trace. Estas perguntas são descritas a seguir.

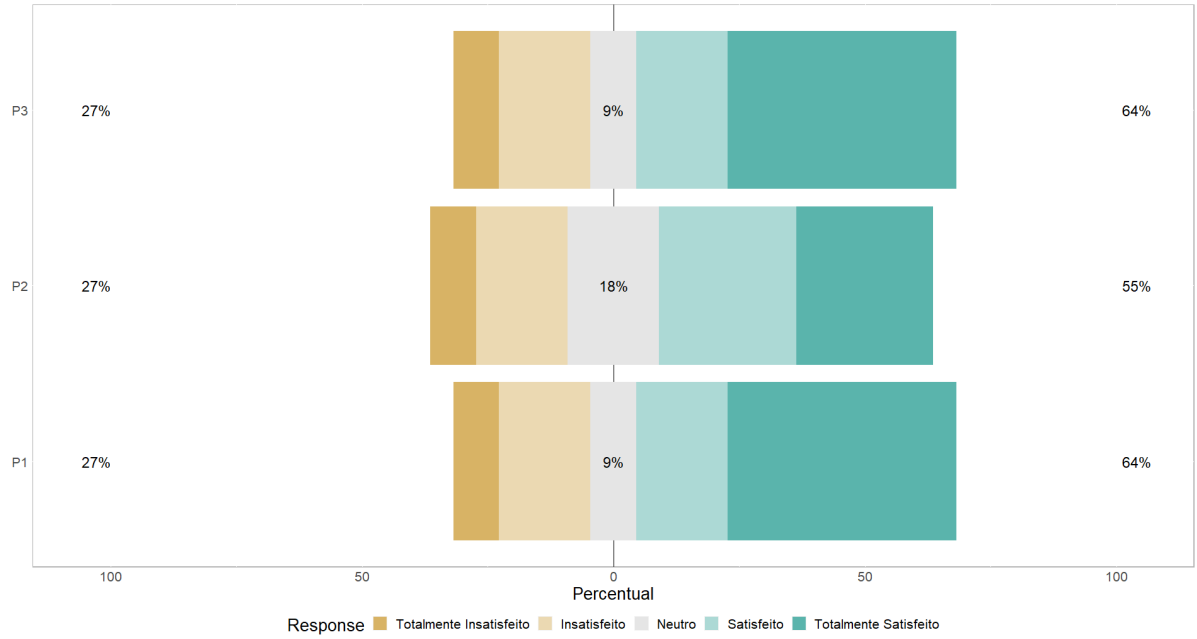


Figura 4.2: Perguntas sobre validação deste trabalho

P1 - Como você avalia sua satisfação com a configurabilidade da ferramenta ?

Esta questão mostra-se direta e pontual para com seu objetivo de verificar a satisfação dos participantes com a configurabilidade da ferramenta. Dado o resultado demonstrado na figura 4.2, mesmo com uma maioria de respostas positivas, com 45,5% de participantes totalmente satisfeitos e 18,2% de satisfeitos, destaca-se um percentual considerável de respostas neutras e negativas. Este percentual é explicado pelo fato de que os participantes não possuíam familiaridade com ferramentas deste tipo, incluso tecnologias que a mesma utilizava.

Esse resultado demonstra que o impacto positivo na configurabilidade foi atingido, mas que este contexto do Feature-Trace ainda apresenta potencial e espaço para melhorias.

P2 - Como você avalia sua satisfação com a usabilidade da ferramenta ?

Nota-se na análise da figura 4.2 um equilíbrio sobre respostas positivas quando comparadas as neutras e negativas para esta pergunta. Mesmo com um percentual maior, essa diferença mínima não pode definir um resultado positivo para esta pergunta. Explica-se este resultado através da reflexão sobre o uso do Feature-Trace por parte dos participantes, que apresentou erros internos para alguns destes.

P3 - Como você avalia sua satisfação com o processo de integração e aplicação da ferramenta em seu ambiente de trabalho?

Dada a análise dos percentuais apresentados na figura 4.2 para as respostas desta questão, tem-se novamente uma maioria positiva em relação à satisfação dos participantes, dessa vez sobre o processo de integração e aplicação da ferramenta. Esta questão resume o processo de configuração, integração e uso da ferramenta e tem como objetivo a validação deste contexto de forma generalizada. Verifica-se também um percentual considerável com respostas neutras e negativas, o que também é explicado pela apresentação de erros internos na utilização do Feature-Trace, além da falta de familiaridade com ferramentas similares.

P8 - Quanto tempo você utilizou para configurar e integrar a ferramenta ao seu ambiente de trabalho ?

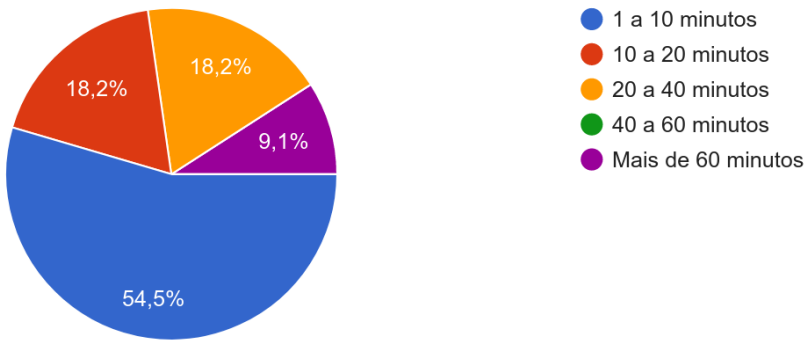


Figura 4.3: Pergunta sobre tempo de execução da ferramenta

Encerrando a categoria de questões sobre a validação de contribuição deste trabalho, esta questão mostra-se mais objetiva e quantificativa ao ter como objetivo a obtenção da métrica de tempo de configuração e integração da ferramenta.

Mesmo com um percentual em sua maioria positivo, demonstrando um simples grau de esforço para com a instalação de ambiente do Feature-Trace através de um curto tempo para tal, verifica-se na figura 4.3 um percentual considerável para tempos acima do ideal. Este resultado é explicado pelo fato de que alguns participantes experienciaram problemas na instalação da ferramenta, como na obtenção de pré requisitos da mesma (artefatos do *Python*) e configuração da ferramenta em um ambiente que apresenta containerização com artefatos do *Docker*. Detalha-se estas dificuldades na próxima seção.

Levantamento de estado atual e utilizade do Feature-Trace

Questões categorizadas neste grupo tem como objetivo o levantamento do estado atual do Feature-Trace, informação crucial para sua evolução. Obtêm-se informações sobre pontos impactados pelas contribuições desse trabalho, além de marcos em que os próximos passos da ferramenta podem seguir e evoluir. Estas questões são descritas a seguir.

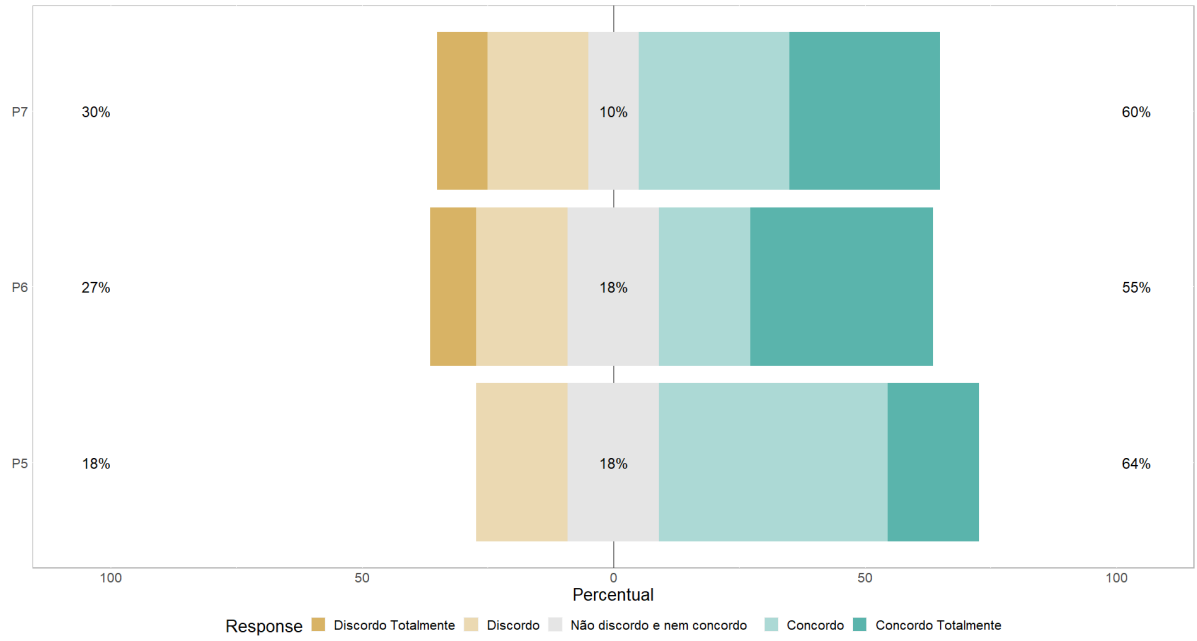


Figura 4.4: Perguntas sobre estado da ferramenta

P5 - Sobre a afirmação: Você utilizaria o Feature-Trace novamente em outra ocasião se precisasse de uma ferramenta similar e com o output que a mesma possui.

Objetivou-se com esta questão o entendimento sobre o interesse dos participantes para com o Feature-Trace. O resultado disponível na figura 4.4 mostra que, mesmo com alguns problemas de execução, os participantes em sua maioria consideraram que a ferramenta possui relevância e que a usariam novamente, mostrando novamente o potencial do Feature-Trace.

P6 - Sobre a afirmação: O Feature-trace possui configurabilidade satisfatória e pode ser usada no cotidiano do desenvolvimento de um software.

Com essa questão, reuniu-se a opinião dos participantes sobre o uso cotidiano e deliberado da ferramenta por parte de um usuário desenvolvedor de *software*. Com uma dife-

rença mínima de percentual apontando um resultado, em sua maioria positivo, nota-se na figura 4.4 que o Feature-Trace ainda precisa de melhorias para que este seja considerado polido o suficiente para sua utilização deliberada, de acordo com o ponto de vista dos participantes.

P7 - Os dados providos pela Feature-Trace contribuíram para melhorar seu entendimento sobre seu projeto?

Almejou-se através desta questão a obtenção da opinião dos participantes sobre a utilidade da ferramenta, dada sua natureza de extração de métricas de um *software* analisado. O resultado em grande parte positivo apresentado na figura 4.4, demonstra e valida o potencial do Feature-Trace.

P9 - Quais das seguintes informações providas na Feature-Trace você considerou mais relevante para seu projeto?

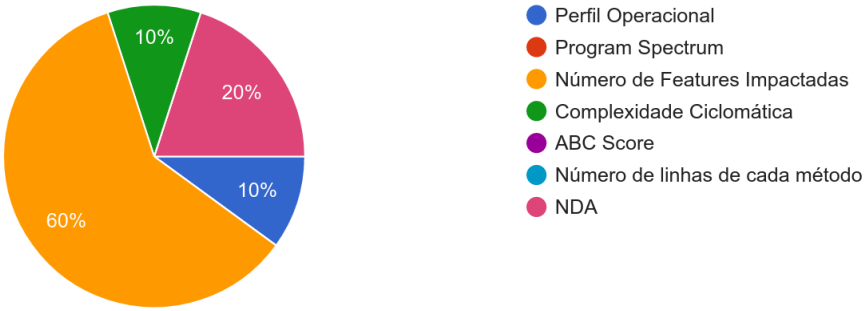


Figura 4.5: Pergunta sobre relevância de métricas

Por fim, apresenta-se outra questão focada em entender a utilidade do Feature-Trace de acordo com o ponto de vista dos desenvolvedores que o testaram. A predominância de respostas para a métrica número de *features* impactadas por entidade é explicada pelo fato de que esta métrica é uma das primeiras apresentadas na análise de um *software* com a ferramenta, obtida na execução de um método simples.

Além disso, verifica-se na figura 4.5 um percentual considerável apontando que para alguns participantes, nenhuma métrica foi relevante. Essa constatação é explicada pelo fato de que alguns destes participantes tiveram erros na análise de seus projetos, dificultando a obtenção de métricas específicas que estes almejavam.

4.3 Lições Aprendidas e Dificuldades Encontradas

Através dos resultados obtidos com o estudo de caso, verificou-se que um certo grau de qualidade na configurabilidade e usabilidade foi alcançado para com o Feature-Trace. Valida-se também essa constatação através de comentários efetuados por parte de alguns participantes, exaltando a configurabilidade, instalação no ambiente local e integração ao projeto analisado.

Sendo assim, o estudo de caso realizado valida que os seguintes resultados esperados descritos na tabela 3.1 foram alcançados: novo fluxo de utilização da ferramenta e módulos com *deployment* e hospedagem remota. O novo fluxo de utilização tornou possível que uma sessão prática com o formato citado fosse executada, além de obter avaliações positivas por parte dos participantes do estudo de caso. Isso só foi possível dada a nova abordagem para com os módulos, agora com hospedagem remota.

Além disso, a distribuição de resultados neutros e negativos nas respostas do questionário expõem espaço para evolução e desenvolvimento, necessárias para que a ferramenta alcance excelência como instrumento de obtenção e análise de métricas de um *software*. Esta necessidade se alinha com um dos objetivos deste trabalho, que é a estruturação e aprimoramento para o fluxo de desenvolvimento de novas funcionalidades. Com as contribuições abordadas por este trabalho considerando o escopo definido, desenvolvedores e contribuidores terão agilidade e qualidade na entrega de valor para o Feature-Trace.

Sobre as dificuldades encontradas, aponta-se a execução do estudo de caso em contexto remoto via vídeo chamada, dificultando o auxílio aos participantes e soluções de dúvidas.

Além disso, citou-se, por parte dos participantes, uma série de dificuldades no teste da ferramenta que prejudicaram a utilização da mesma. Os problemas são detalhados a seguir:

- Dificuldades na integração do Feature-Trace em projetos que possuem *Docker* (containerização).
- Execuções de alguns métodos de análise do Feature-Trace que apresentaram bloqueios, impedindo a obtenção de métricas almejadas.

Naturalmente, incluir a ferramenta em um ambiente que apresenta containerização traz complicações para sua configuração, dado que pré requisitos e o processo de instalação deve ser abordado e definido por artefatos de configuração do *Docker*, como citado na seção 3.4. Esse tipo de configuração, que deve ser construída pelo desenvolvedor do projeto, não é idealmente elaborada durante uma sessão prática, dado os gargalos apresentados.

Em relação aos erros de execução internos do Feature-Trace, os mesmos estão sendo apresentados em alguns casos onde as entidades analisadas abordam algum tipo de inte-

ração com o usuário. Este tipo de erro interno deve ser registrado e corrigido brevemente, dado o intuito de manter o interesse e relevância da ferramenta.

Evidencia-se nas limitações encontradas na execução do Feature-Trace um desafio e problema: a integração da ferramenta em projetos com ambientes e configurações distintas. Os problemas na integração de projetos com containerização e erros em métodos que abordam interação com usuário inferem este entendimento, sendo que este desafio acompanhará a evolução da ferramenta em contribuições futuras.

Estas dificuldades citadas pelos participantes, em conjunto com a falta de conhecimento e familiaridade com este tipo de ferramenta, explica alguns resultados divididos e assim não conclusivos para as perguntas do questionário enviado aos mesmos. De qualquer modo, o descobrimento sobre tais dificuldades é benéfico para a ferramenta, dado que estas serão envolvidas e correlacionadas à próximas contribuições para a ferramenta.

Capítulo 5

Conclusão

A principal implementação e contribuição deste projeto pode ser associada à adoção de conceitos do *DevOps* para com o Feature-Trace. Com essa adoção, aprimorou-se a ferramenta em ambas perspectivas de desenvolvimento e do usuário final.

Obteve-se um novo fluxo de desenvolvimento para o Feature-Trace, que agora apresenta práticas e técnicas que auxiliam na elaboração de contribuições, principalmente no quesito de agilidade e garantia de qualidade na integração de código ao *software* existente.

No escopo de *Deployment*, os módulos cliente e servidor que compõem a ferramenta passaram a possuir soluções de hospedagem *online*. A entrega de valor, manifestada através da entrega de *software* possui um fluxo mais natural e ágil a partir dessa nova abordagem.

Na perspectiva do usuário, a adoção do *DevOps* e *deployment* para com o Feature-Trace possibilitou a formação de um novo fluxo de utilização, simplificando a configurabilidade e usabilidade da ferramenta e removendo barreiras para sua utilização deliberada.

Como resultado de todas as contribuições e objetivos auxiliares deste trabalho, a ferramenta passou a utilizar técnicas e conceitos comumente utilizados pela indústria e comunidade de desenvolvimento de *software*, alinhamento benéfico para o ciclo de vida do Feature-Trace como *software* de código aberto e que contribui para a composição de trabalhos futuros.

Para a validação das contribuições executadas por este trabalho, principalmente na esfera do uso do usuário final, elaborou-se um estudo de caso em forma de sessão prática. Através deste estudo, verificou-se que um certo grau de qualidade na configurabilidade e usabilidade foi alcançado para com o Feature-Trace. Além disso, nota-se também que a ferramenta precisa de contribuições para sua estabilização e evolução para que a mesma alcance um estado de excelência.

Para trabalhos futuros, sugere-se a estabilização da ferramenta, refletindo sobre a correção de problemas na execução da mesma evidenciados no estudo de caso deste trabalho.

No quesito de implementação de novas funcionalidades para a ferramenta, aconselha-se a elaboração de um módulo para que o Feature-Trace possa analisar projetos desenvolvidos em linguagens distintas do *Ruby*, como o *Python*. Além disso, seria benéfico para a ferramenta contribuições e aprimoramento em seu servidor, como melhorias na experiência de usuário e em sua interface. Por fim, sugere-se também uma validação da ferramenta em um contexto industrial, dada que a mesma teve validação apenas em ambiente acadêmico.

Referências

[1] Fazzolino, R. e G. N Rodrigues: *Feature-trace: An approach to generate operational profile and to support regression testing from bdd features.* In Proceedings of the XXXIII Brazilian Symposium on Software Engineering, páginas 332—-336, 2019. ix, 1, 7, 8, 9

[2] Wynne, Hellesoy, Tooke: *The cucumber book: behaviour-driven development for testers and developers*, volume 2. Pragmatic Bookshelf, 2017. 2, 5

[3] Beck, Kent: *Test-driven development: by example.* Addison-Wesley Professional, 2003. 5

[4] *Gherkin.* `https://cucumber.io/docs/gherkin/`, acesso em 2021-10-05. 5

[5] Smart, John Ferguson: *BDD in Action.* Manning Publications, 2014. 6

[6] Musa, J. D.: *The operational profile in software reliability engineering: an overview. in proceedings third international symposium on software reliability engineering.* IEEE Computer Society, páginas 140—-141, 1992. 6, 7, 9

[7] Harrold, Mary Jean, Gregg Rothermel, Rui Wu e Liu Yi: *An empirical investigation of program spectra.* SIGPLAN Not., 33:83—-90, 1998. 7

[8] McCabe, T.J.: *A complexity measure.* IEEE Transactions on Software Engineering, (4):308–320, 1976. 9

[9] Fitzpatrick, Jerry: *Applying the ABC Metric to C, C++, and Java*, página 245–264. Cambridge University Press, 2000. 9

[10] Paraiso, Fawaz, Stéphanie Challita, Yahya Al-Dhuraibi e Philippe Merle: *Model-driven management of docker containers.* Em *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*, páginas 718–725, 2016. 10

[11] Dua, Rajdeep, A. Reddy Raja e Dharmesh Kakadia: *Virtualization vs containerization to support paas.* 2014 IEEE International Conference on Cloud Engineering, páginas 610–614, 2014. 10

[12] Koskinen, Mikael, Tommi Mikkonen e Pekka Abrahamsson: *Containers in software development: A systematic mapping study.* Em Franch, Xavier, Tomi Männistö e Silverio Martínez-Fernández (editores): *Product-Focused Software Process Improvement*, páginas 176–191, Cham, 2019. Springer International Publishing. 10

[13] Erich, F. M. A., C. Amrit e M. Daneva: *A qualitative study of devops usage in practice.* Journal of Software: Evolution and Process, 29(6):e1885, 2017. e1885 smr.1885. 10

[14] Smeds, Jens, Kristian Nybom e Ivan Porres: *Devops: A definition and perceived adoption impediments.* Em Lassenius, Casper, Torgeir Dingsøyr e Maria Paasivaara (editores): *Agile Processes in Software Engineering and Extreme Programming*, páginas 166–177, Cham, 2015. Springer International Publishing. 11

[15] Kromhout, Bridget: *Containers will not fix your broken culture (and other hard truths).* Commun. ACM, 61(4):40–43, março 2018. 11

[16] Luz, Welder Pinheiro, Gustavo Henrique Lima Pinto e Rodrigo Bonifácio: *Adopting devops in the real world: A theory, a model, and a case study.* J. Syst. Softw., 157, 2019. 11, 13, 14, 17

[17] Fowler, Martin e Matthew Foemmel: *Continuous integration*, 2006. 12

[18] Savor, Tony, Mitchell Douglas, Michael Gentili, Laurie Williams, Kent Beck e Michael Stumm: *Continuous deployment at facebook and oanda.* Em *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)*, páginas 21–30, 2016. 12

[19] Lewis, James e Martin Fowler: *Microservices: a definition of this new architectural term.* MartinFowler. com, 25:14–26, 2014. 12

[20] *Github*. `https://github.com`, acesso em 2021-10-19. 15

[21] *Python*. `https://www.python.org/`, acesso em 2021-10-23. 15

[22] *Django*. `https://www.djangoproject.com/`, acesso em 2021-10-17. 15

[23] *Postgresql*. `https://www.postgresql.org/`, acesso em 2021-10-17. 15

[24] Nichols, David M e Michael B Twidale: *Usability and open source software.* 2002. 15

[25] Neus, Andreas e Philipp Scherf: *Opening minds: Cultural change with the introduction of open-source collaboration methods.* IBM Systems Journal, 44(2):215–225, 2005. 16

[26] *Docker*. `https://www.docker.com/`, acesso em 2021-10-18. 19

[27] *Docker compose*. `https://docs.docker.com/compose/`, acesso em 2021-10-19. 21

[28] *Github action*. `https://github.com/features/actions`, acesso em 2021-10-19. 23, 29

[29] *Heroku*. `https://www.heroku.com`, acesso em 2021-10-19. 25

[30] Humble, J., C. Read e D. North: *The deployment production line.* Em *AGILE 2006 (AGILE'06)*, páginas 6 pp.–118, 2006. 25

[31] *Click*. `https://click.palletsprojects.com`, acesso em 2021-10-22. 25

[32] Rao, Vaddina Prakash e Dimitri Marandin: *Adaptive backoff exponent algorithm for zigbee (ieee 802.15.4)*. Em Koucheryavy, Yevgeni, Jarmo Harju e Villy B. Iversen (editores): *Next Generation Teletraffic and Wired/Wireless Advanced Networking*, páginas 501–516, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg, ISBN 978-3-540-34430-8. 26

[33] *Pypi*. `https://pypi.org/`, acesso em 2021-10-23. 30

[34] *Trace feature*. `https://pypi.org/project/trace-feature`, acesso em 2021-10-23. 31, 34

Anexo I

Questionário sobre utilização do Feature Trace

Questões que utilizam escala Likert: satisfação

- P1 - Como você avalia sua satisfação com a configurabilidadeda ferramenta ?
- P2 - Como você avalia sua satisfação com a usabilidade da ferramenta ?
- P3 - Como você avalia sua satisfação com o processo de integração e aplicação da ferramenta em seu ambiente de trabalho? ?

Opções:

- 1 - Totalmente Insatisfeito
- 2 - Insatisfeito
- 3 - Neutro
- 4 - Satisfeito
- 5 - Totalmente Satisfeito

Questões que utilizam escala Likert: probabilidade

- P4 - Quando você pensa em utilizar uma ferramenta, avalia o quão trabalhoso será a integração da mesma em seu ambiente.

Opções:

- 1 - Nunca
- 2 - Raramente

- 3 - Ocasionalmente
- 4 - Frequentemente
- 5 - Sempre

Questões que utilizam escala Likert: Concordância

- P5 - Sobre a afirmação: Você utilizaria o Feature Trace novamente em outra ocasião se precisasse de uma ferramenta similar e com o output que a mesma possui.
- P6 - Sobre a afirmação: O Feature-trace possui configurabilidade satisfatória e pode ser usada no cotidiano do desenvolvimento de um software.
- P7 - Os dados providos pela Feature Trace contribuíram para melhorar seu entendimento sobre seu projeto?

Opções:

- 1 - Discordo Totalmente
- 2 - Discordo
- 3 - Não discordo nem concordo
- 4 - Concordo
- 5 - Concordo Totalmente

Questões finais: tempo de configurabilidade e artefatos do Feature-Trace

- P8 - Quanto tempo você utilizou para configurar e integrar a ferramenta ao seu ambiente de trabalho ?

Opções:

- 1 a 10 minutos
- 10 a 20 minutos
- 20 a 40 minutos
- 40 a 60 minutos
- Mais de 60 minutos

- P9 - Quais das seguintes informações providas na Feature Trace você considerou mais relevante para seu projeto?

Opções:

- Perfil Operacional
- Program Spectrum
- Número de Features Impactadas
- Complexidade Ciclomática
- ABC Score
- Número de linhas de cada método
- NDA