# Simplificação do Processo de Configuração de uma Pipeline de Integração e Entrega Contínua no Jenkins

**Christyan Santos Rosa[1], Lucas Maffra Vieira[1], Lesandro Ponciano[1]**

[1]Bacharelado em Engenharia de Software
Instituto de Ciências Exatas e Informática – PUC Minas
Rua Cláudio Manoel, 1.162, Funcionários, Belo Horizonte – MG – Brasil

{christyanrosa, lucas.maffra}@sga.pucminas.br, lesandrop@pucminas.br

***Abstract.*** *The increase of agile methodologies in the software development market makes it necessary to provide fast and continuous deliveries of software increments, without losing quality. Therefore, to ensure those deliveries with quality, it is necessary to implement a process of integration and continuous delivery, but such process is often complicated to understand and set up. This article intent is to improve the process of setting up a pipeline in Jenkins application, so that it's simpler to understand. It proposes a simplification strategy and shows the obtained gain through it in terms of ease of configuration of the pipeline. The results show improvement in the usability of the application.*

*key-words: Continuous Integration, Continuous Delivery, Jenkins, Pipeline*

***Resumo.*** O aumento do uso de metodologias ágeis no mercado de desenvolvimento de software acarreta o aumento da necessidade mais necessário entregas rápidas e contínuas de incrementos de software, sem prejuízo para a qualidade do software. Para garantir essas entregas rápidas e com qualidade é necessário a implantação de um processo de integração e entrega contínua, mas, tal processo geralmente é complicado de ser entendido e configurado por usuários novatos ou pouco frequentes. Este artigo tem por objetivo simplificar o processo de configuração de uma *pipeline* de modo que ele seja mais simples de ser entendido por usuários novatos ou pouco frequentes. Por meio de uma avaliação heurística são investigados os problemas de usabilidade nesse processo na ferramenta Jenkins. Como estudo de caso, propõe-se uma estratégia de simplificação da ferramenta Jenkins e mostra-se o ganho obtido em termos de simplicidade de configuração da *pipeline*. Os resultados mostram melhoria na usabilidade da ferramenta.

*Palavras chave: Integração Contínua, Entrega Contínua, Jenkins, Pipeline*

## 1. Introdução

Devido ao crescimento no desenvolvimento de novas tecnologias e suas complexidades, as metodologias de desenvolvimento tradicionais são pouco resilientes para se adaptarem rapidamente às mudanças. Há um recorrente atraso nas entregas, o cliente não consegue visualizar o resultado de imediato e validar se o que foi desenvolvido está de acordo com suas expectativas [Petrochina 2018]. Para possibilitar mais flexibilidade,

eficiência e velocidade durante o ciclo de vida de um software surgiram as metodologias ágeis e com elas a integração contínua (CI do inglês *continuous integration*) e entrega contínua (CD do inglês *continuous delivery*) [Arachchi and Perera 2018] [Eddy et al. 2017]. Os projetos que utilizam integração contínua melhoram a qualidade do código em 50% e tornam mais simples a detecção e correção de *bugs* em cerca de 65% [Arachchi and Perera 2018].

O processo de CI e CD contribui com o desenvolvimento ágil, pois sua *pipeline* automatiza os testes, *build* e empacotamento do software [Eddy et al. 2017] [Arachchi and Perera 2018]. Apesar de ser consolidado no mercado, o aprendizado dessa área se mostra complexo devido à quantidade de ferramentas disponíveis e suas curvas de aprendizado [Eddy et al. 2017]. Apesar dos desenvolvedores conseguirem utilizar o processo de CI/CD no seu cotidiano, a configuração inicial da *pipeline* pode se tornar uma tarefa complicada, principalmente para aqueles desenvolvedores novatos ou que não fazem tal configuração com frequência, assim como Eddy et al. (2017) mostrou. **A dificuldade da configuração inicial da *pipeline* na ferramenta Jenkins é, portanto, o problema tratado neste estudo.**

É importante ressaltar que o processo de gestão da configuração de software contribui para o sucesso de projetos, garante a qualidade do projeto e facilita o gerenciamento de versão [Rocha et al. 2007]. A gestão da configuração de software começa antes mesmo do seu desenvolvimento e segue atuando até a entrega final. Esse processo garante que em cada nova implementação, mudança de configuração e ambiente de execução, o software continua funcionando como esperado. É necessário para a utilização desse processo que os envolvidos se adaptem e compreendam como é o seu funcionamento, porque é um processo desafiante principalmente devido ao aumento da complexidade dos softwares.

Tendo em vista o contexto apresentado, **o objetivo deste trabalho é propor uma estratégia para simplificar a configuração inicial de uma *pipeline* de CI/CD por usuários novatos ou pouco frequentes**. Para atingir esse objetivo, são propostos os seguintes objetivos específicos: Desenvolver uma estratégia para simplificação da *pipeline* de CI/CD; analisar o impacto da utilização dessa estratégia; entregar uma solução que se mostre simples de ser utilizada e que, portanto, contribua para configuração inicial da *pipeline*.

Por fim, neste estudo pode-se citar duas principais contribuições. Primeiro, a criação de uma interface de configuração de *pipeline* que é mais simples de utilizar para usuários novatos ou pouco frequentes. Segundo, o mapeamento dos problemas que os usuários do Jenkins enfrentam ao realizar uma configuração de *pipeline*.

O trabalho está organizado em 7 seções. Na seção 2 está a fundamentação teórica. Nela é possível compreender todos os conceitos para a leitura da pesquisa. A seção 3 contempla os trabalhos relacionados. Já a seção 4 possui os materiais e métodos utilizados. Na seção 5 é apresentado a proposta de solução. A seção 6 apresenta os resultados. Por fim a seção 7 com as conclusões e trabalhos futuros.

## 2. Fundamentação Teórica

Nesta seção, são apresentados os conceitos necessários para a compreensão de uma *pipeline* de CI/CD. Entender o funcionamento desse processo é essencial para este estudo, assim como o funcionamento da integração contínua, entrega contínua e seu funcionamento no *framework* Jenkins.

### 2.1 DevOps

Integração e entrega contínua surgem no contexto atualmente chamado de DevOps ou Release Engineering [Ebert et al. 2016]. O DevOps é um termo criado da junção das palavras desenvolvimento (do inglês *development*) e operações (do inglês *operations*). Ele representa uma metodologia de desenvolvimento de software que utiliza a comunicação para integrar desenvolvedores de software e profissionais de infraestrutura de TI. Essa metodologia ficou conhecida como implementação contínua ou entrega contínua porque, ao padronizar ambientes de desenvolvimento, também auxilia no gerenciamento do lançamento de novas versões. DevOps é a integração eficiente do desenvolvimento, entrega e operações de forma a tornar esses diferentes processos bem conectados e fluidos.

### 2.2 Integração Contínua e Entrega Contínua

A integração contínua é uma abordagem utilizada pela equipe de desenvolvimento que permite integrar constantemente os novos códigos e implementações ao sistema, por meio de automatizações de *builds*, testes e validações [Adams and McIntosh 2016][Arachchi and Perera 2018]. É importante salientar que o processo de CI possibilita entrega menores e mais frequentes e aumenta a produtividade da equipe [Adams and McIntosh 2016][Shahin, Ali and Zhu 2017][Humble and Farley 2010].

A Figura 1 mostra o ciclo básico de CI que começa quando o desenvolvedor introduz uma nova alteração no repositório de controle de versão, o servidor processa os testes automatizados e em seguida realiza *build* do projeto. Caso ocorra algum erro durante os testes ou *build*, o desenvolvedor recebe um *feedback* indicando que ele precisa fazer alguma alteração e submetê-la ao repositório reiniciando o ciclo [Eddy et al. 2017].

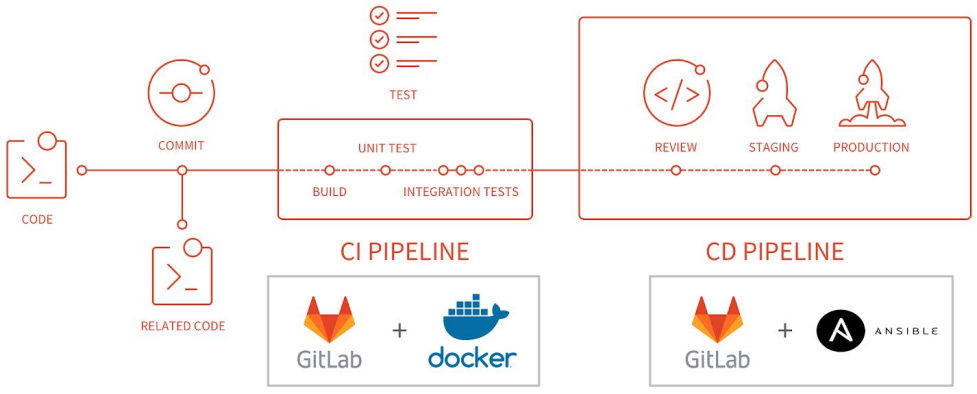

**Figura 1. Ciclo de CI/CD**

**Fonte: https://docs.gitlab.com/ce/ci/**

Os estágios do processo de CI/CD são retratados na Figura 1 como *review*, *staging* e *production* que representam, respectivamente, o estágio onde o código sofre validação estática, o estágio onde ocorre o *deploy* para um ambiente onde a qualidade do software e suas funcionalidades são testados e o estágio do *deploy* para produção. O GitLab trata-se de um repositório de código pronto para CI/CD. GitLab possui uma ferramenta de CI/CD integrada, mas também possui integração com ferramentas como Jenkins, para criação e configuração da *pipeline* de CI/CD, o Docker, ferramenta para criação e gerenciamento de contêineres, e o Ansible, ferramenta que automatiza o *deploy* dos contêineres e administra os *clusters* e seus nós [Humble and Farley 2010].

O processo de CD faz parte do processo de CI e ele consiste em entregar um novo pacote de alterações que inclui novos recursos e correções de *bugs* de forma segura e rápida garantindo que as mudanças não afetaram o sistema em produção [Arachchi and Perera 2018]. Os princípios de CD podem ser utilizados em diversos sistemas, tais como sistemas de larga escala, sistemas embarcados ou até mesmo softwares comerciais. O importante é conseguir disponibilizar alterações de forma rápida, segura e automaticamente para o ambiente de produção[Humble 2018].

Utilizar CD no desenvolvimento de software dá vantagem competitiva para as organizações. Ela permite às equipes disponibilizarem as funcionalidades quando estão concluídas dando mais estabilidade e resiliência ao sistema. Contudo, algumas empresas ainda possuem restrições a esse processo dizendo que: CD é inadequado para trabalhar em ambientes regulamentados; a entrega contínua é apenas para *websites*; não podem ser aplicadas em sistemas legados; a entrega contínua requer profissionais com mais experiência e talento do que os disponíveis na empresa, o que contraria os princípios de CI/CD [Humble 2018].

## 2.3 Jenkins

O Jenkins é um *framework* de integração contínua de código aberto, desenvolvido na linguagem de programação Java. Ele é amplamente difundido no mercado para a construção e implantação de software, pois permite verificação constante do sistema de gerenciamento de versão, roda os testes automatizados e notifica aos desenvolvedores a situação do processo de CI [Sampedro, Holt and Hauser 2018]. Apesar do seu foco no processo de construção de software, a ferramenta também disponibiliza formas de ser estendida, através de diversos *plugins* desenvolvidos pela comunidade [Moutsatsos et al. 2017].

O Jenkins possibilita que a importação de uma *pipeline* seja feita por meio de dois tipo de arquivos, XML ou *jenkinsfile*. O XML trata-se de um arquivo de marcação onde cada seção e campos da configuração ficam descritos. O *jenkinsfile*, é um formato da própria aplicação, onde seu formato indica cada passo da *pipeline* e sua configuração, a Figura 2 mostra um exemplo de *jenkinsfile*.

```
Jenkinsfile
pipeline {
    stages {
        stage('Build') {
            steps {
                echo 'Building..'
            }
        }
        stage('Test') {
            steps {
                echo 'Testing..'
            }
        }
        stage('Deploy') {
            steps {
                echo 'Deploying....'
            }
        }
    }
}
```

**Figura 2. Exemplo de *jenkinsfile***

## 3. Trabalhos Relacionados

Nesta seção, são apresentados os trabalhos relacionados à construção de uma *pipeline* de CI/CD para simplificação do processo de desenvolvimento de software. Contudo, nem todos tratam exclusivamente da elaboração inicial da *pipeline,* mas apresentam abordagens relevantes a este estudo.

Eddy et al. (2017b) propõem a criação de uma *pipeline* para utilização no meio acadêmico para ensino do processo de CI/CD. O objetivo do trabalho é mostrar, a novos desenvolvedores, o ganho do processo e uma forma simples de criação e configuração da *pipeline*. Diferentemente de Eddy et al (2017b), este trabalho foca exclusivamente em profissionais não estudantes e busca-se uma forma automatizada ou de suporte ao processo de configuração inicial de CI/CD. Assim, o trabalho conduzido neste estudo segue a linha de simplificar a criação e configuração de tal processo, de forma que ele seja aplicável a qualquer projeto de software que utilize CI/CD.

Em um segundo estudo, Eddy et al. (2017a) buscaram identificar os níveis de experiência e conhecimento dos participantes com as tarefas do processo de CD e se, após o estudo de caso, os alunos têm um maior entendimento sobre as tarefas e conceitos de CI. Os resultados obtidos mostram que é necessário um foco em conceitos de teste unitário e análise estática do código. Parte da dificuldade de entender o processo está na baixa compreensão desses conceitos.

Arachchi e Pereira (2018) abordam as práticas de CI/CD em sistemas escaláveis que necessitam de alta performance, para isso utiliza as ferramentas GoReplay, Nagios, Jenkins, Git e Nexus. Os autores constroem uma *pipeline* e mostram os efeitos da adição de testes de carga para melhorar a entrega do software. A conclusão é que ao acrescentar essa etapa no processo há um ganho na qualidade da entrega e na produtividade. O trabalho citado mostra a adição de um item na *pipeline* de CI/CD. Diferentemente desse estudo, o intuito deste trabalho é contribuir com a etapa inicial da elaboração desse processo, permitindo que mais itens sejam adicionados sem aumentar a barreira de configuração da *pipeline*.

Em seu estudo PetroChina (2018) constrói uma *pipeline* de CI com uma rede topológica conectada com vários servidores, com o intuito de verificar os benefícios da CI em projetos de software. O autor mostra que CI/CD diminuem consideravelmente o tempo das entregas. O estudo em questão avalia o desempenho de CI/CD. Nesta pesquisa o objetivo é contribuir para o desempenho da elaboração do processo, para que fique mais simples para os envolvidos compreenderem o seu funcionamento e o configure adequadamente.

De acordo com Sampedro, Holt e Hauser (2018), sistemas computacionais de alta performance necessitam constantemente de atualizações e para manter o desenvolvimento é necessário a utilização de ferramentas que mantenham constante validação sobre o que é implementado na aplicação. A proposta deste trabalho não está na otimização do processo, mas na redução da barreira ao seu uso.

## 4. Materiais e Métodos

Esta pesquisa tem caráter exploratório e qualitativo. Possui característica exploratória por se tratar de uma análise do fluxo de configuração de um CI/CD no Jenkins e identificar quais passos o usuário que for utilizar a nova abordagem precisa compreender e preencher. A pesquisa é qualitativa porque consiste na avaliação da facilidade dos desenvolvedores que realizaram a configuração pela interface padrão do Jenkins em comparação com a estratégia proposta. Nesta avaliação, são empregados métodos de inspeção, observação e de investigação, respectivamente. O restante desta seção apresenta os procedimentos para a realização deste estudo, assim como os instrumentos utilizados.

### 4.1. Procedimentos

O estudo parte da análise da situação atual do Jenkins para desenvolver uma ferramenta para simplificar o processo de configuração de CI/CD no contexto do *framework* Jenkins. Primeiramente, faz-se o mapeamento das variáveis para a configuração da *pipeline*. Posteriormente, com os dados sobre os potenciais usuários da ferramenta, desenvolve-se um ambiente que viabilize a configuração da *pipeline* e gere um arquivo de configuração que possa ser utilizado no Jenkins.

### 4.2. Instrumento de avaliação por observação

A avaliação por observação realizado em outubro de 2019, contou com a participação de três profissionais de desenvolvimento de empresas do setor público e dois do setor privado, localizadas em Belo Horizonte e por dois alunos do curso de Engenharia de Software da PUC Minas. No início da realização do experimento explicitou-se para os participantes uma síntese do experimento e foi entregue um documento contendo as informações necessárias para a configuração da *pipeline*. Para contextualização, elaborou-se em formato de história para não haver favorecimento da configuração em nenhuma abordagem. A seguir está a história de construção da *pipeline*:

- Você como desenvolvedor trabalha em um projeto de desenvolvimento software, que possui o seu código versionado no repositório GitHub, a URL do projeto é

https://github.com/ChristyanS/experimento-jenkins. Foi solicitado que você configure uma *pipeline* de integração e entrega contínua para melhorar a qualidade e velocidade das entregas deste software. A *pipeline* que você deve configurar deve conter as validações de qualidade de código, que pode ser feito com o seguinte comando "mvn sonar:sonar-Dsonar.projectKey=calculadora-Dsonar.host.url=http://localhost:9000-Dsonar.login=admin -Dsonar.password=admin". Em seguida, a execução dos testes é realizada com "mvn test". Após as validações de código e testes, é necessário a realização da *build* do projeto, isso deve ser feito com o comando "mvn compile". Em seguida realizar o empacotamento do projeto com o comando "mvn package". Por fim, fazer o deploy com o comando "mvn deploy".

Durante a execução do experimento utilizou-se um formulário para armazenamento das informações sobre a execução do observado, como: os horários de início e fim da realização da configuração; qual interface estava sendo utilizada no experimento pelo observado, nativa do Jenkins ou estratégia proposta; se possui experiência com CI/CD; quais perguntas foram realizadas pelo observado e as respostas providas pelos observadores.

### 4.3. Instrumento de avaliação por investigação

A avaliação por investigação ocorreu após a realização da avaliação por observação com o mesmo grupo de participantes da avaliação por observação. Mostra-se importante a avaliação da ferramenta desenvolvida, para conseguir mensurar os impactos proporcionados por ela. Buscou-se criar dois grupos que representam uma amostra heterogênea dos usuários. O primeiro grupo faz a configuração manual do Jenkins na versão 2.203 e o segundo faz na ferramenta proposta. Posteriormente ambos responderam um questionário padrão, o *System Usability Scale* [Brooke 2013]. Para a avaliação de ambas as abordagens, propõe-se que os participantes do experimento realizem um mesmo conjunto de atividades, o que permite utilizar métricas para comparar a usabilidade das estratégias e o desempenho dos desenvolvedores ao utilizá-las [Barbosa and Silva 2010].

A escala de usabilidade do sistema (SUS do inglês *system usability scale*) é um questionário baseado na Escala Likert, no qual os participantes devem demonstrar seu nível de concordância com a afirmação. O SUS foi concebido por John Brookes em 1986 quando realizava testes de usabilidade no laboratório da *Digital Equipament Corporation* e percebeu que os métodos que utilizava possuíam um grau de complexidade alto e tempo elevado para execução [Brookes 2013]. O questionário do SUS é utilizado após a realização de um teste de usabilidade no sistema avaliado. Ele é composto por 10 perguntas como mostra a Tabela 1, nas quais para cada uma delas o participante deve responder em uma escala de 1 a 5, onde 1 significa “Discordo Completamente” e 5 significa “Concordo Completamente” [Brookes 2013].

**Tabela 1. Perguntas do questionário SUS**

| N | Item |
|---|---|
| 1 | Eu acho que gostaria de usar esse sistema com frequência |

| | |
|---|---|
| 2 | Eu acho o sistema desnecessariamente complexo |
| 3 | Eu achei o sistema fácil de usar. |
| 4 | Eu acho que precisaria de ajuda de uma pessoa com conhecimentos técnicos para usar o sistema. |
| 5 | Eu acho que as várias funções do sistema estão muito bem integradas. |
| 6 | Eu acho que o sistema apresenta muita inconsistência. |
| 7 | Eu imagino que as pessoas aprenderão como usar esse sistema rapidamente. |
| 8 | Eu achei o sistema atrapalhado de usar. |
| 9 | Eu me senti confiante ao usar o sistema. |
| 10 | Eu precisei aprender várias coisas novas antes de conseguir usar o sistema. |

Após a coleta das respostas é realizado um cálculo para cada item do formulário. As respostas com números ímpares (1,3,5,7,9) tem seu valor subtraído de 1, já as respostas com números pares (2,4,6,8,10) o valor da resposta é subtraído de 5. Depois de calcular os *scores* individuais os valores são multiplicados por 2,5 e somados. A pontuação encontrada no final é o resultado da avaliação de usabilidade que pode variar de 0 a 100 pontos, tendo valor de referência de 68 pontos.

### 4.4. Instrumento de avaliação por inspeção

A avaliação por inspeção inicialmente foi realizada pelos autores para identificar problemas na interface do Jenkins. Após a implementação da interface proposta foi realizada a avaliação na interface proposta e na interface do Jenkins por um novo grupo de avaliação formado por 3 alunos, do curso de Engenharia de Software da PUC Minas, para verificar se os problemas encontrados na interface do Jenkins correspondiam com os problemas encontrados pelos autores e que os selecionados para correção não contam na interface proposta.

A avaliação por inspeção trata-se de um avaliador colocando-se no lugar do usuário e realizando uma inspeção da aplicação, utilizando por exemplo a avaliação de heurística. A avaliação heurística permite a avaliação de uma solução sem a utilização dos usuários reais, pois ela tem o intuito de tentar prever as possíveis consequências das decisões de design. Os responsáveis por avaliar são pessoas com experiência e conhecimento acerca das atividades realizadas, o que possibilita a verificação de problemas [Barbosa and Silva 2010].

Dentre os métodos de avaliação por inspeção, selecionou-se a avaliação heurística, desenvolvida por Nielsen e Molich em 1990 para processos de design iterativos, com o intuito de verificar problemas de usabilidade. A vantagem de utilizá-lo é devido ao baixo custo e tempo para execução se comparado aos empíricos. Este método usa como base heurísticas de Nielsen criadas para agrupar diretrizes de usabilidade. As heurísticas são: visibilidade do estado do sistema; correspondência entre o sistema e o mundo real; controle e liberdade do usuário ; consistência e padronização; reconhecimento ao invés de memorização; flexibilidade e eficiência de uso; projeto

estético e minimalista; prevenção de erros; ajuda os usuários a reconhecer, diagnosticar e recuperar de erros; ajuda e documentação [Barbosa and Silva 2010].

Selecionou-se para participar do estudo alunos da PUC da graduação em Engenharia de Software e profissionais da área de desenvolvimento com experiência em CI/CD. Os alunos selecionados já haviam cursado as disciplinas de Gerência de Configuração e Evolução de Software, que se refere ao conhecimento sobre CI/CD, e Interação Humano-Computador, que refere ao conhecimento sobre avaliação heurística na análise de interfaces.

## 5. Proposta de nova interface

Nesta seção são apresentados os passos realizados para a construção da nova abordagem da configuração inicial. Descreve-se o usuário alvo considerado na construção do sistema, análise da situação atual do Jenkins e um comparativo entre ambas as abordagens. A simplificação é a criação de uma interface interativa e com elementos de visibilidade de estado do sistema. Nessa interface, o usuário recebe informações da etapa da configuração da *pipeline*, enquanto preenche-as, e após o preenchimento, a estratégia gera a *pipeline* configurada e que pode ser importada diretamente para o Jenkins.

### 5.1 Usuário Alvo

Devido ao problema principal tratado neste artigo, mapeou-se o usuário da abordagem com um perfil que teria dificuldade para configurar uma *pipeline* de CI/CD. O público alvo do sistema desenvolvido são desenvolvedores que possuem conhecimento sobre os conceitos de CI/CD, entretanto inexperientes na configuração do processo. São pessoas que utilizam *pipelines* já configuradas ou realizaram somente a configuração de uma de forma didática. Essas pessoas são consideradas usuários novatos ou pouco frequentes na configuração de *pipeline* de CI/CD.

### 5.2 Análise da situação atual e solução

Ao realizar análise do funcionamento da interface do Jenkins, utilizou-se um método de avaliação por inspeção conduzida pelos autores deste trabalho. Utilizou-se para avaliação o Jenkins versão 2.203 e foi analisado somente a tela de criação de novo *job*. Durante a avaliação mapeou-se cada heurística violada, o local da ocorrência, justificativa e solução como mostra a Tabela 2. A avaliação resultou em um total de 7 violações que afetam a utilização do sistema, que podem ser agrupadas em: Visibilidade do Estado do Sistema, Controle e Liberdade do Usuário, Projeto estético e minimalista, Reconhecimento ao invés de memorização e Prevenção de Erros.

**Tabela 2. Avaliação Heurística no Jenkins**

| Heurística | Problema | Solução |
|---|---|---|
| Visibilidade do Estado do Sistema | Informações não são caracterizadas de forma que o usuário consiga visualizar cada etapa de configuração de uma *pipeline* do projeto analisado por ele. | Construção de um indicador de passos que contém as etapas de CI/CD permitindo o usuário visualizar em qual passo da *pipeline* ele está: *repository*, *quality*, *test*, *package*, *deploy*. |

| | | |
|---|---|---|
| Correspondência entre o sistema e o mundo real | Não foi encontrado nenhum problema | Não foi criada nenhuma solução |
| Controle e Liberdade Para o Usuário | Ao clicar em "Avançado...", como em descartar *builds* antigos, mais campos são exibidos e não é possível retornar para a visualização anterior com apenas os campos que lá estavam. Isso se torna um problema em telas em que abrem-se diversos campos e que não podem mais ser ocultados. | Permitir reverter as ações, voltando para o estado original. |
| Consistência e padronização | O idioma da aplicação está mesclado português e inglês. Isso se torna um problema, pois o usuário pode não compreender o que deve ser feito | Padronizar o idioma |
| Reconhecimento em vez de memorização | Não foi encontrado nenhum problema | Não foi criada nenhuma solução |
| Flexibilidade e eficiência de uso | Não foi encontrado nenhum problema | Não foi criada nenhuma solução |
| Projeto estético e minimalista | A tela de configurações possui diversos campos e todos estão dispostos em uma única página. Apesar de apresentar uma quantidade baixa de campos ao iniciar a configuração (cerca de 24 campos para *jobs Freestyle*), a medida em que o usuário, que está realizando a configuração, preenche os campos da tela, ela passa a conter uma alta quantidade de campos, podendo chegar a mais de 75 campos sem a utilização de *plugins*.<br>A maioria dos campos mostrados possui uma dica explicando qual o intuito daquela configuração específica, mas os textos das dicas podem ser longos e eles aparecem abaixo da informação selecionada, expandindo o tamanho da página. Por exemplo o *checkbox* "Este *build* é parametrizado" que possui um texto educativo de 15 linhas e 6 parágrafos.<br>Ao expandir ou minimizar campos de *TextArea* os demais componente podem deixar de ser exibidos totalmente na tela e não é possível reverter a ação. | Divisão da tela de acordo com as etapas de configuração da *pipeline* para diminuir a quantidade de campos exibidos na tela por vez, exibindo somente campos da etapa que ele está configurando.<br><br>Disponibilização de dicas dinâmicas que permita o usuário visualizar de forma simplificada a dica na tela, ou seja, ao clicar na dica ela fica sobreposta e destacada do resto da página, permitindo uma informação pontual. |
| Prevenção de erros | É possível criar novos *jobs* sem nenhum conteúdo. Isso se torna um problema já que o usuário pode criar uma *pipeline* que não conclui o processo de CI/CD. | Obrigatoriedade dos campos. |

| | | |
|---|---|---|
| Ajude os usuários a reconhecer, diagnosticar e recuperar de erros | Não foi encontrado nenhum problema | Não foi criada nenhuma solução |
| Ajuda e documentação | Não foi encontrado nenhum problema | Não foi criada nenhuma solução |

Considerando os problemas identificados, foram selecionados para desenvolvimento da nova interface os seguintes itens: o problema de identificação da etapa da configuração que está efetivamente sendo configurada; a falta de possibilidade de reverter as alterações realizadas na tela de configuração de *pipeline*, como a adição de novos campos de configurações; a apresentação de todos os campos simultâneos na tela de configuração.

### 5.3 Interface que soluciona os problemas

Elaborou-se um protótipo de alta fidelidade utilizando o Adobe XD para validar se o artefato gerado caracterizava-se por conter um fluxo de interação que solucionava cada um dos problemas de heurísticas detectados. Após a criação do protótipo desenvolveu-se uma aplicação Web externa ao Jenkins que permite integração por meio da criação de um *jenkinsfile*. A interface de configuração do Jenkins é desenvolvida para atender diversos tipos de *scripts* de automação, é possível criar configurações nos seguintes formatos: *free-style*; *pipeline*; construir projeto com múltiplas configurações; *folder*; GitHub Organization; *multibranch pipeline*. Devido a esta característica as configurações criadas no Jenkins não seguem necessariamente a ordem de configuração uma *pipeline* (*repository*, *quality, test, package, build* e *deploy*).

É possível visualizar na Figura 3 que a ferramenta categoriza suas informações em 6 tipos, já a abordagem proposta Figura 4 possui um indicador de passos que permite o usuário visualizar exatamente as etapas de construção da *pipeline*, o que facilita o entendimento acerca dos conceitos de CI/CD. A interface do Jenkins também caracteriza-se por conter uma quantidade elevada de campos, que foram detectadas durante a fase de avaliação, estas informações são apresentadas em uma única página e dificulta o preenchimento das informações. A abordagem desenvolvida subdivide as informações entre as categorias de configuração da *pipeline* o que diminui a quantidade de campos exibidos para o usuário por vez.

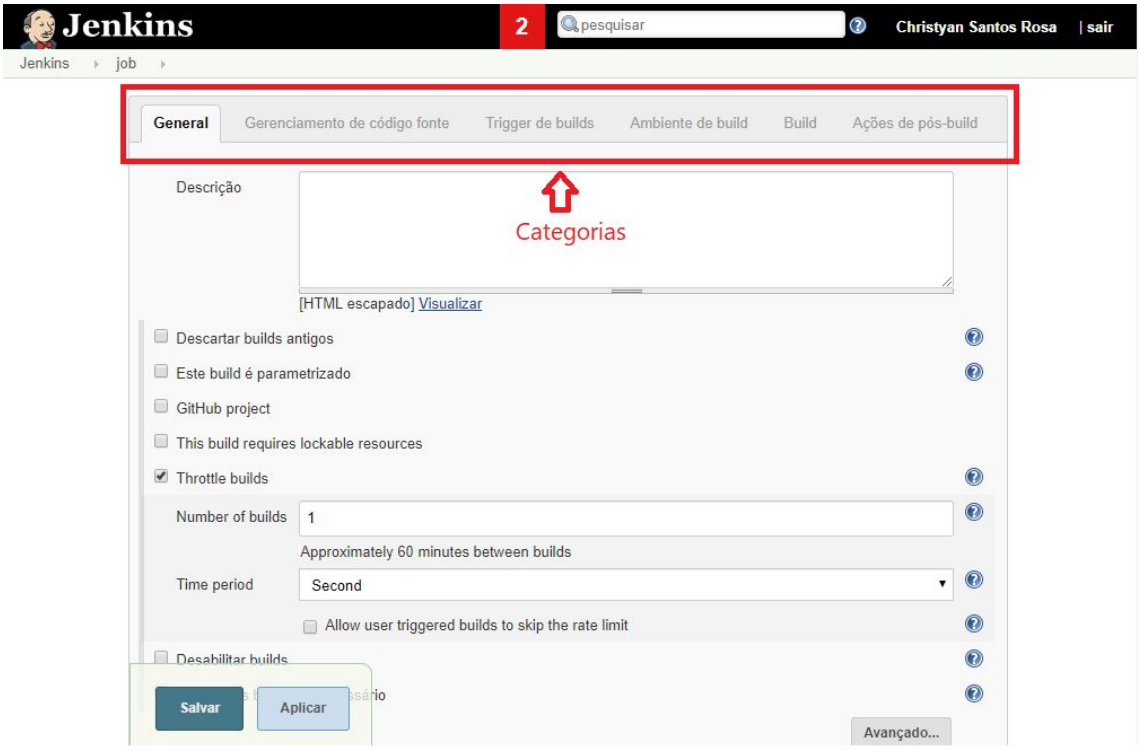


**Figura 3. Tela de configurações no Jenkins**

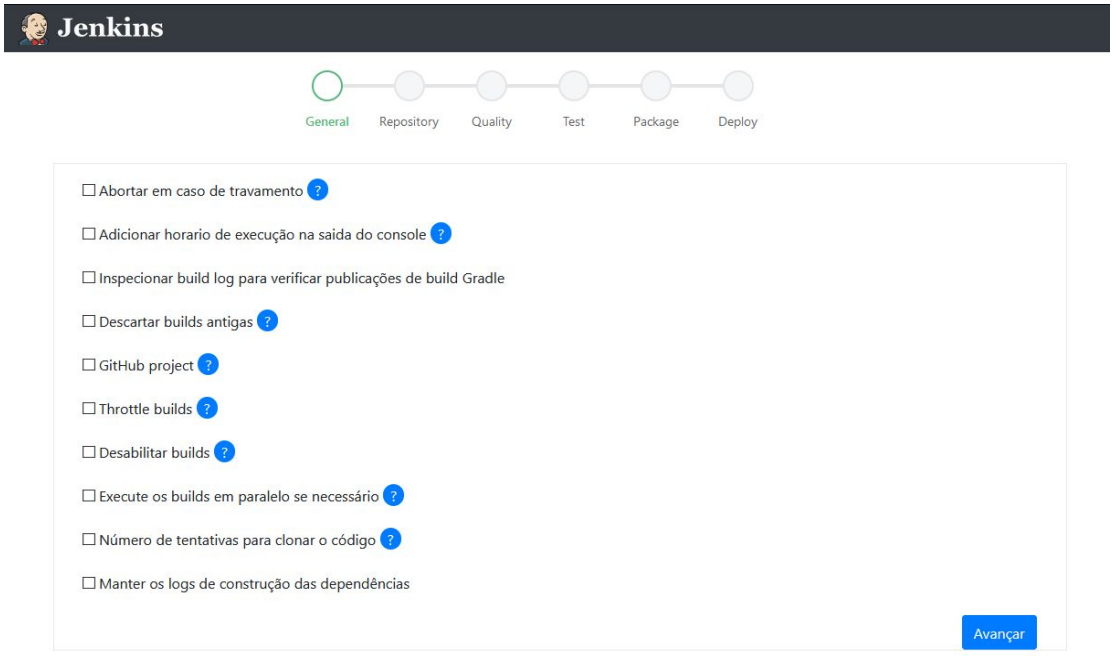


**Figura 4. Tela de Configuração com a abordagem proposta**

## 6. Resultados

Nesta seção, apresenta-se os resultados obtidos na aplicação do método proposto para identificar se a implementação proposta reduz a dificuldade na configuração de uma *pipeline* de CI/CD. A seção é dividida em 3 partes. Na primeira são apresentados os

resultados encontrados na avaliação por inspeção. Na segunda os resultados encontrados na avaliação por observação. Por fim, na terceira estão os resultados encontrados na avaliação por investigação.

### 6.1 Resultado da avaliação por inspeção

A avaliação por inspeção gerou um resultado consolidado das heurísticas violadas mostrado na Tabela 3. Observou-se que o total de heurísticas violadas no Jenkins é superior a interface proposta, sendo que o Jenkins possui maior quantidades de violações de grau cosmético. A nova interface apresenta uma redução 33,33% em violações catastróficas e 60% nas graves.

Das heurísticas que tiveram soluções para os problemas é perceptível que para a heurística Visibilidade do Estado do Sistema houve uma redução das violações e da severidade, que era de 2 erros graves e agora há 1 violação simples. Já na heurística Controle e Liberdade para o Usuário teve um aumento no número de problemas que era de 1 cosmético e agora são de 2 cosméticos, 1 simples e 1 catastrófico, no qual o catastrófico não há obrigatoriedade dos campos, problema não tratado por este trabalho. Por fim para a heurística Projeto Estético e Minimalista teve uma redução de 1 violação cosmética.

**Tabela 3. Violações das Heurísticas em ambas abordagens**

| Heurística | Jenkins | | Proposta | |
|---|---|---|---|---|
| Visibilidade do Estado do Sistema | **Grau** | **Quantidade** | **Grau** | **Quantidade** |
| | Sem importância | 0 | Sem importância | 0 |
| | Cosmético | 0 | Cosmético | 0 |
| | Simples | 0 | Simples | 1 |
| | Grave | 2 | Grave | 0 |
| | Catastrófico | 0 | Catastrófico | 0 |
| Correspondência entre o sistema e o mundo real | **Grau** | **Quantidade** | **Grau** | **Quantidade** |
| | Sem importância | 0 | Sem importância | 0 |
| | Cosmético | 0 | Cosmético | 0 |
| | Simples | 1 | Simples | 0 |
| | Grave | 0 | Grave | 0 |
| | Catastrófico | 0 | Catastrófico | 0 |
| Controle e Liberdade Para o Usuário | **Grau** | **Quantidade** | **Grau** | **Quantidade** |
| | Sem importância | 0 | Sem importância | 0 |
| | Cosmético | 1 | Cosmético | 2 |
| | Simples | 0 | Simples | 1 |
| | Grave | 0 | Grave | 0 |
| | Catastrófico | 0 | Catastrófico | 1 |
| Consistência e padronização | **Grau** | **Quantidade** | **Grau** | **Quantidade** |
| | Sem importância | 2 | Sem importância | 0 |
| | Cosmético | 3 | Cosmético | 3 |
| | Simples | 3 | Simples | 1 |
| | Grave | 1 | Grave | 0 |
| | Catastrófico | 1 | Catastrófico | 0 |

| | | | | |
|---|---|---|---|---|
| Reconhecimento em vez de memorização | **Grau** | **Quantidade** | **Grau** | **Quantidade** |
| | Sem importância | 0 | Sem importância | 0 |
| | Cosmético | 0 | Cosmético | 0 |
| | Simples | 0 | Simples | 0 |
| | Grave | 0 | Grave | 0 |
| | Catastrófico | 0 | Catastrófico | 0 |
| Flexibilidade e eficiência de uso | **Grau** | **Quantidade** | **Grau** | **Quantidade** |
| | Sem importância | 0 | Sem importância | 0 |
| | Cosmético | 3 | Cosmético | 0 |
| | Simples | 0 | Simples | 2 |
| | Grave | 1 | Grave | 2 |
| | Catastrófico | 0 | Catastrófico | 0 |
| Projeto estético e minimalista | **Grau** | **Quantidade** | **Grau** | **Quantidade** |
| | Sem importância | 1 | Sem importância | 1 |
| | Cosmético | 2 | Cosmético | 1 |
| | Simples | 0 | Simples | 0 |
| | Grave | 0 | Grave | 0 |
| | Catastrófico | 0 | Catastrófico | 0 |
| Prevenção de erros | **Grau** | **Quantidade** | **Grau** | **Quantidade** |
| | Sem importância | 0 | Sem importância | 0 |
| | Cosmético | 0 | Cosmético | 0 |
| | Simples | 0 | Simples | 0 |
| | Grave | 0 | Grave | 0 |
| | Catastrófico | 1 | Catastrófico | 1 |
| Ajude os usuários a reconhecer, diagnosticar e recuperar de erros | **Grau** | **Quantidade** | **Grau** | **Quantidade** |
| | Sem importância | 0 | Sem importância | 0 |
| | Cosmético | 0 | Cosmético | 0 |
| | Simples | 1 | Simples | 1 |
| | Grave | 0 | Grave | 0 |
| | Catastrófico | 0 | Catastrófico | 0 |
| Ajuda e documentação | **Grau** | **Quantidade** | **Grau** | **Quantidade** |
| | Sem importância | 0 | Sem importância | 0 |
| | Cosmético | 2 | Cosmético | 0 |
| | Simples | 1 | Simples | 1 |
| | Grave | 1 | Grave | 0 |
| | Catastrófico | 1 | Catastrófico | 0 |
| Total | **Grau** | **Quantidade** | **Grau** | **Quantidade** |
| | Sem importância | 3 | Sem importância | 1 |
| | Cosmético | 11 | Cosmético | 6 |
| | Simples | 6 | Simples | 7 |
| | Grave | 5 | Grave | 2 |
| | Catastrófico | 3 | Catastrófico | 2 |

## 6.2 Resultado da avaliação por observação

Na avaliação por observação todos os participantes conseguiram completar a configuração. O tempo médio para realização da configuração da *pipeline* no Jenkins foi

de 21 minutos, com desvio padrão de 13,65 minutos. Na interface proposta esse tempo é de 8,66 minutos, com desvio padrão de 1,29 minutos. Isso significa uma redução de 61,15% no tempo médio de configuração.

Durante o processo de avaliação os participantes realizaram as seguintes perguntas sobre o ambiente: se o *plugin* do Maven estava instalado no jenkins; se era necessária autenticação para conectar ao repositório em ambas interfaces. Na avaliação com a interface do Jenkins, houve questionamentos sobre o que aconteceu com a configuração já inserida que foi apagada após trocar um campo da tela de configuração.

### 6.3 Resultado da avaliação por investigação

Após a avaliação por observação houve a realização da avaliação por investigação com a resposta do SUS pelos participantes. A média da pontuação de cada SUS respondido para a interface do Jenkins foi de 32,5 pontos, com desvio padrão de 12,33 pontos, e para a interface proposta foi de 86,5 pontos, com desvio padrão de 4,26, conforme mostra a Figura 5. Observou-se que a interface proposta teve uma nota melhor que o Jenkins em todos os itens avaliados, como mostra a Figura 6.

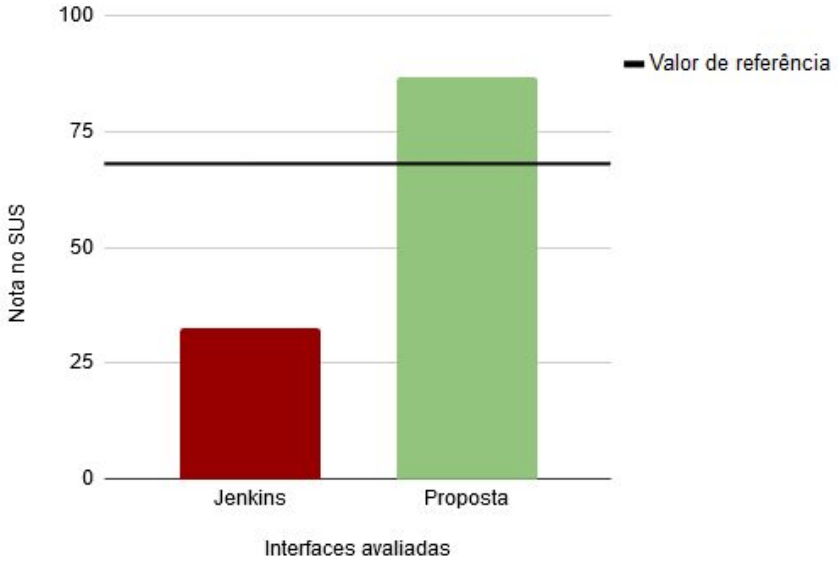


**Figura 5. Resultado das pontuações totais do SUS para a interface do Jenkins e a proposta, linha com a nota média do SUS**

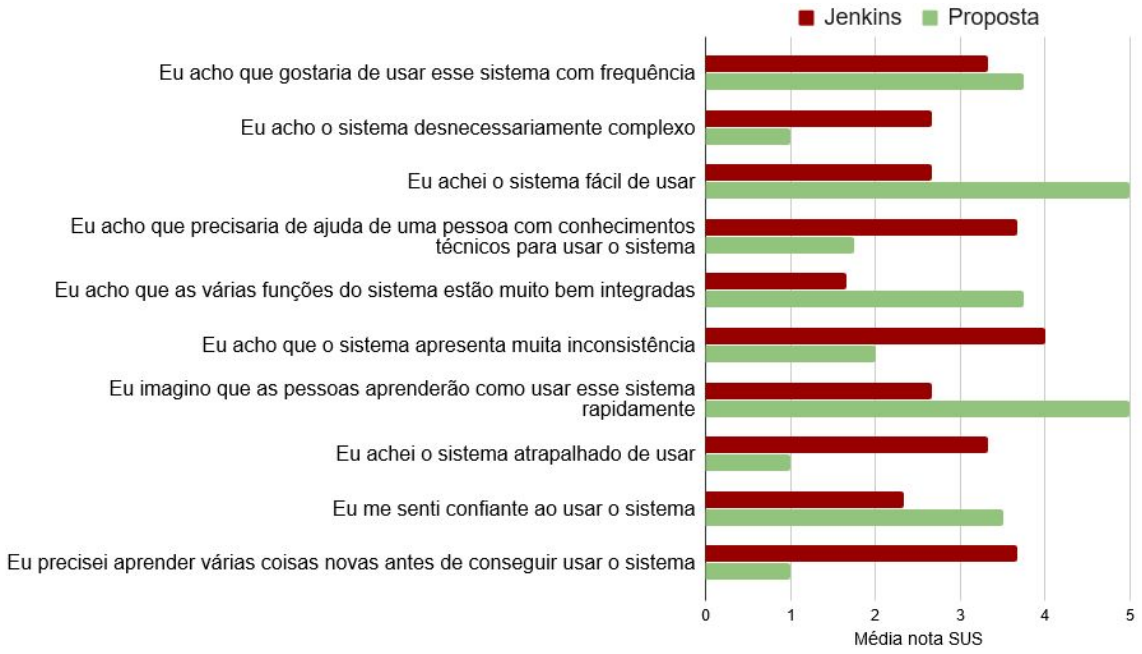


**Figura 6. Resultado das pontuações média do SUS por questão para a interface do Jenkins e a proposta**

## 7. Conclusão e trabalhos futuros

Neste estudo foi proposta uma estratégia com o foco em diminuir a dificuldade da configuração inicial de uma *pipeline* de CI/CD e os pontos observados na aplicação da estratégia foram: o entendimento de cada desenvolvedor sobre a configuração, com as dúvidas que participantes tiveram durante a avaliação por observação; a capacidade de completar a configuração corretamente; as dificuldades que os observados teriam em relação a interface; o tempo para realizar a configuração.

Os resultados mostram que nas duas interfaces os desenvolvedores tiveram sucesso em completar a configuração da pipeline, porém houve uma diminuição na dificuldade de configuração da *pipeline* utilizando a interface proposta. O tempo médio para realização da configuração da *pipeline* foi menor e houve menos questões levantadas sobre a configuração durante o experimento de observação na interface proposta. Além da interface proposta apresentar correções para os problemas identificados como violações de heurísticas na interface do Jenkins, os resultados da avaliação por investigação ressaltam que a interface proposta possui uma usabilidade melhor em relação ao Jenkins.

Para os trabalhos futuros, seria relevante a migração da interface proposta neste artigo para Jenkins, utilizando o Apache Jelly, ou em forma de um *plugin* para que a solução fique dentro da própria aplicação. Após a implementação desta migração, também seria relevante a execução do experimento, adaptado para essa nova solução, para verificar os ganhos na migração. O código com a implementação da proposta está disponível no GitHub em https://github.com/ChristyanS/christyans.github.io.

## Referências


Adams B. and McIntosh S. (2016). Modern Release Engineering in a Nutshell -- Why Researchers Should Care. IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering. p 78-90.

Arachchi, S. A. I. B. S. and Perera, I. (2018). Continuous Integration and Continuous Delivery Pipeline Automation for Agile Software Project Management. *2018 Moratuwa Engineering Research Conference (MERCon)*, p. 156–161.

Barbosa, S. D. J. and Silva, B. S. Da (2010). Interação Humano Computador. 1. ed. Rio de Janeiro: .

Brooke, J. (2013). SUS: A Retrospective. Journal of usability Studies, v. 8, n. 2, p. 29–40.

Ebert, C., Gallardo, G., Hernantes, J. and Serrano, N. (2016). DevOps. IEEE Software, v. 33, n. 3, p. 94–100.

Eddy, B. P., Wilde, N., Cooper, N. A., Mishra B., Gamboa V.S, Shah K.M, Deleon A. M. and Shields N. A. (2017a). A Pilot Study on Introducing Continuous Integration and Delivery into Undergraduate Software Engineering Courses. IEEE Conference on Software Engineering Education and Training, p. 47–5.

Eddy, B. P., Wilde, N., Cooper, N. A., Gamboa V. S, Mishra B., Patel K. N. and Shah K. M. (2017b). CDEP : Continuous Delivery Educational Pipeline ∗. In Proceedings of the SouthEast Conference, ACM SE '17, p. 55–62

Humble, J. (2018). Delivery Sounds Great , but Will It Work Here ?. Communications of the ACM, v. 61, p. 34–39.

Humble, J and Farley, D. (2010). Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation (1st ed.). Addison-Wesley Professional.

Jenkins (2019). Jenkins. https://jenkins.io/ [accessed on Jun 6].

Moutsatsos, I. K., Hossain, I., Agarinis, C. (2017). Jenkins-CI , an Open-Source Continuous Integration System , as a Scientific Data and Image-Processing Platform. Society for Laboratory Automation and Screening, v. 22, p. 238–249.

Petrochina, W. (2018). Design and Implementation of Continuous Integration scheme Based on Jenkins and Ansible. *2018 International Conference on Artificial Intelligence and Big Data (ICAIBD)*, p. 245–249.

Rocha, F. G., Menezes, P. M., Nascimento, R. P. C. Do, Junior, M. C. R. and Oliveira, A. A. De (2007). Continuous Integration and Version Control: a Systematic Review. Journal of Latex Class Files, v. 6, n. 1, p. 1–5.

Sampedro, Z., Holt, A. and Hauser, T. (2018). Continuous Integration and Delivery for HPC Using Singularity and Jenkins. PEARC '18: Practice and Experience in Advanced Research Computing, p. 4–6.

Shahin, M., Ali, M. and Zhu, L. (2017). Continuous Integration , Delivery and

Deployment : A Systematic Review on Approaches , Tools , Challenges and Practices. IEEE.