UNIVERSIDADE DO ESTADO DA BAHIA (UNEB)
DEPARTAMENTO DE CIÊNCIAS EXATAS E DA TERRA - CAMPUS I
CURSO DE BACHARELADO EM SISTEMAS DE INFORMAÇÃO

PEDRO VICTOR SANTANA BENEVIDES

BIODOCKFLOW: UM PROCESSO DE MANUTENÇÃO BASEADO EM CONTÊINERES DOCKER PARA APLICAÇÕES WEB HETEROGÊNEAS DE BIOINFORMÁTICA

SALVADOR
2024

PEDRO VICTOR SANTANA BENEVIDES

BIODOCKFLOW: UM PROCESSO DE MANUTENÇÃO BASEADO EM CONTÊINERES DOCKER PARA APLICAÇÕES WEB HETEROGÊNEAS DE BIOINFORMÁTICA

Monografia apresentada ao Curso de Bacharelado em Sistemas de Informação do Departamento de Ciências Exatas e da Terra (DCET) - Campus I, da Universidade do Estado da Bahia (UNEB), como requisito à obtenção do grau de bacharel em Sistemas de Informação.
Área de concentração: Engenharia de Software

Orientador: Prof. Dr. Alexandre Rafael Lenz

SALVADOR
2024

PEDRO VICTOR SANTANA BENEVIDES

BIODOCKFLOW: UM PROCESSO DE MANUTENÇÃO BASEADO EM CONTÊINERES DOCKER PARA APLICAÇÕES WEB HETEROGÊNEAS DE BIOINFORMÁTICA

Monografia apresentada ao Curso de Bacharelado em Sistemas de Informação do Departamento de Ciências Exatas e da Terra (DCET) - Campus I, da Universidade do Estado da Bahia (UNEB), como requisito à obtenção do grau de bacharel em Sistemas de Informação.
Área de concentração: Engenharia de Software

Aprovada em: 17/12/2024.

BANCA EXAMINADORA

Prof. Dr. Alexandre Rafael Lenz
Orientador

Profa. Dra. Mônica de Souza Massa
Examinador interno (DCET-I/UNEB)

Prof. Dr. Vagner de Souza Fonseca
Examinador interno (DCET-I/UNEB)

AGRADECIMENTOS

A Deus, por me dar obstáculos e me fazer mais forte. A minha família, que sempre foi a minha base. A minha mãe, Raidalva Santana, por todas as conquistas que me permitiram chegar até aqui. Ao meu orientador, Prof. Dr. Alexandre Rafael Lenz, por todo apoio, orientação e companheirismo no desenvolvimento deste trabalho. Aos professores desta gradução pela minha formação.

"Uma mentalidade de campeão. Não se trata de vencer, nem de troféus – trata-se de nunca desistir e de dar tudo o que você tem."

(Chris Bumstead)

RESUMO

A bioinformática, enquanto campo interdisciplinar que integra biologia e computação, desempenha um papel central na pesquisa científica e na inovação tecnológica. Essa área tem transformado a compreensão dos sistemas biológicos ao combinar dados complexos com análises computacionais avançadas, por meio de uma ampla gama de aplicações web. Muitos desses softwares são disponibilizados gratuitamente à comunidade científica, potencializando avanços e alcançando objetivos cada vez mais específicos. Contudo, a diversidade de tais aplicações apresenta desafios significativos relacionados à manutenibilidade e interoperabilidade, os quais impactam diretamente a reprodutibilidade das pesquisas científicas. Neste contexto, este trabalho tem como objetivo propor um processo para a manutenção de aplicações web de bioinformática, fundamentado na utilização de contêineres Docker, adotando a metodologia *Design Science Research* (DSR). O processo desenvolvido, denominado BioDockFlow, foi aperfeiçoado iterativamente em conformidade com os princípios da DSR através de três ciclos, sendo submetido a avaliações quantitativas, por meio da definição de uma métrica, e qualitativa em seu terceiro e último ciclo, com a aplicação de um questionário direcionado a usuários do processo. Os resultados obtidos corroboraram as conjecturas teóricas delineadas na metodologia, destacando que, embora processos de manutenção estruturados e controlados apresentem eficácia e adaptabilidade, essas características podem ser comprometidas pela heterogeneidade das aplicações web de bioinformática. Nesse contexto, o BioDockFlow foi capaz de orientar os desenvolvedores na conteinerização de aplicações com diferentes níveis de complexidade, garantindo a interoperabilidade entre elas em um ambiente unificado e atendendo às demandas específicas de cada aplicação. Ademais, o processo demonstrou ser uma ferramenta eficaz no suporte a equipes de manutenção, promovendo a padronização das atividades, facilitando o alinhamento das ações realizadas e identificando oportunidades de aprimoramento. Ressalta-se, ainda, a capacidade do BioDockFlow de integrar melhorias práticas, evidenciando sua adequação aos princípios de melhoria contínua. O processo está disponível no repositório do G2BC no GitHub[1].

Palavras-chave: Aplicações web; Bioinformática; Docker; Processo de Manutenção.

[1] https://github.com/G2BC/BioDockFlow

ABSTRACT

Bioinformatics as an interdisciplinary field integrating biology and computing, plays a central role in scientific research and technological innovation. This domain has transformed the understanding of biological systems by combining complex data with advanced computational analyses through a wide range of web-based applications. Many of these software tools are freely available to the scientific community, fostering progress and enabling increasingly specific objectives. However, the diversity of such applications presents significant challenges related to maintainability and interoperability, which directly impact the reproducibility of scientific research. In this context, this study aims to propose a systematic process for maintaining bioinformatics web applications, grounded in the use of Docker containers and employing the Design Science Research (DSR) methodology. The proposed process, named BioDockFlow, was iteratively refined through three cycles in alignment with DSR principles. It underwent quantitative evaluations based on defined metric and qualitative assessments in its third and final cycle through a structured questionnaire administered to users of the process. The results obtained corroborated the theoretical conjectures outlined in the methodology, highlighting that, although structured and controlled maintenance processes demonstrate effectiveness and adaptability, these characteristics may be challenged by the heterogeneity of bioinformatics web applications. In this context, BioDockFlow proved capable of guiding developers in the containerization of applications with varying levels of complexity, ensuring interoperability among them within a unified environment and meeting the specific demands of each application. Furthermore, the process demonstrated its efficacy as a tool for supporting maintenance teams by promoting activity standardization, facilitating action alignment, and identifying improvement opportunities. Notably, BioDockFlow's ability to integrate practical enhancements underscores its alignment with the principles of continuous improvement.The process is available in the G2BC repository on GitHub[2].

Key-words: Web application; Bioinformatics; Docker; Maintenance process.

[2] https://github.com/G2BC/BioDockFlow

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

UNEB	Universidade do Estado da Bahia
DSR	Design Science Research
G2BC	Grupo de Pesquisa em Bioinformática e Biologia Computacional
IVET	Influenza Viral Evolution Tool
PLASTICOME	Fungal Genome Plastic Degradation Explorer
BEM	Brazilian Edible Mushrooms
HTML	HyperText Markup Language
CSS	Cascading Style Sheets
JS	Javascript
CSG	CIPRES Science Gateway
CMMI	Capability Maturity Model Integration
SEI	Software Engineering Institute
QM	Questão de Manutenção
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
URL	Uniform Resource Locator

SUMÁRIO

1 INTRODUÇÃO

Softwares de bioinformática enfrentam complexos ciclos de vida devido a desafios relacionados com escalabilidade, portabilidade e manutenção. Em muitos casos, pesquisadores almejam distribuir software científico como parte de projetos de pesquisa, mas carecem de recursos adequados em engenharia de software para manter um servidor de aplicações web robusto capaz de lidar com a coexistência de múltiplos softwares (Tam *et al.*, 2023). Essa limitação dificulta o acesso às aplicações além de comprometer a replicabilidade e evolução das pesquisas mais difícil, o que, em última instância, pode reduzir o impacto científico do projeto (Boettiger, 2015).

A coexistência de softwares de bioinformática representa um desafio significativo de interoperabilidade, especialmente em ambientes onde diversas aplicações heterogêneas precisam ser executadas em um mesmo servidor (Silver, 2017). Para lidar com essa complexidade, é essencial implementar processos e métodos que garantam que cada aplicação possa operar de forma independente, sem afetar negativamente o desempenho ou a funcionalidade das outras (Sansone *et al.*, 2012).

Além disso, para promover uma interoperabilidade robusta, é fundamental considerar a virtualização de recursos e ambientes. Através da virtualização, é possível criar ambientes isolados para cada aplicação, garantindo que elas possam coexistir de forma segura e eficiente no mesmo servidor físico. Essa estratégia reduz o risco de conflitos entre os softwares e simplifica a administração e a manutenção do ambiente, permitindo uma maior flexibilidade na gestão dos recursos de computação (Boettiger, 2015). Portanto, investir em estratégias de interoperabilidade avançadas não apenas facilita a execução harmoniosa de múltiplos softwares de bioinformática, mas também contribui para uma pesquisa mais eficaz e colaborativa.

Conferências de engenharia de software já apresentaram preocupação com a replicação (Cito; Gall, 2016), sinalizando a importância da reprodutibilidade no campo. Esse fator pode ser ainda mais aprimorado se todos os artefatos pertencentes aos softwares que são apresentados em artigos forem empacotados e documentados em contêineres, permitindo que pesquisadores realizem o uso imediato da aplicação sem problemas de dependências. Nesse âmbito, ao encapsular todo o ambiente computacional necessário para executar um experimento ou análise em um contêiner Docker, os pesquisadores podem garantir que outras iniciativas científicas possam reproduzir seus resultados de forma consistente, independentemente do sistema operacional e das configurações de software. Isso elimina tanto a preocupação em afetar o funcionamento de outros softwares no mesmo ambiente quanto a necessidade de lidar com problemas de compatibilidade e dependência, tornando a replicação e a evolução de pesquisas científicas muito mais simples e confiável (Cito;

Gall, 2016).

O presente projeto de pesquisa integra uma iniciativa do G2BC, que atualmente opera um servidor Docker na UNEB. Em consulta realizada com os docentes do grupo, constatou-se que as aplicações web desenvolvidas pelo G2BC apresentam características heterogêneas, ou seja, diversificam-se em termos de linguagens de programação, arquiteturas, interação com serviços adicionais, bancos de dados, pipelines implementados e gerenciamento de dependências. Esses softwares precisam coexistir e operar de forma independente no mesmo servidor. Diante desse cenário, torna-se essencial a implementação da virtualização em nível de contêiner com Docker, visando não apenas atender ao requisito do servidor disponível, mas também contemplar as características singulares de manutenção e identificação de novos requisitos específicos de cada aplicação, conforme destacado pelos docentes.

Diante disso, levantou-se a questão de pesquisa norteadora deste projeto: *Um processo de manutenção baseado em Docker pode contribuir para manutenção das aplicações web de bioinformática assegurando interoperabilidade?*

Tendo como base a questão problema, o objetivo geral deste projeto tem como finalidade propor um processo para manutenção de aplicações web de bioinformática baseado em conteinerização utilizando Docker. Para atingir esse objetivo foi necessário atingir os seguintes objetivos específicos:

- utilizar as principais referências da área de engenharia de software para identificar as etapas necessárias para criar um processo de manutenção;
- incluir no processo proposto os diferentes tipos de manutenção de software encontrados na literatura para atender necessidades específicas de manutenção de cada aplicação;
- validar, de forma quantitativa e qualitativa, o processo proposto por meio da implementação em aplicações web de bioinformática com variados níveis de complexidade e heterogeneidade, utilizando contêineres para a realização do *deploy*.

Espera-se como resultados, após avaliar de forma quantitativa e qualitativa, o atendimento às necessidades de manutenção de conteinerização Docker dos softwares do G2BC assegurando a interoperabilidade das aplicações no mesmo servidor. Adicionalmente, projeta-se contribuir para manutenção de aplicações web de bioinformática de forma que o processo facilite a identificação de necessidades de mudanças e refatoração de código.

Este trabalho de conclusão de curso está estruturado em seis capítulos. O capítulo 2 apresenta a fundamentação teórica necessária para o desenvolvimento do projeto, abordando temas como aplicações web de bioinformática, interoperabilidade, Docker, ciclo de vida

de software, processos de manutenção e trabalhos relacionados. O capítulo 3 trata da metodologia de pesquisa adotada. O capítulo 4 especifica o processo desenvolvido e os resultados obtidos nos ciclos de iteração. O capítulo 5 apresenta a validação do processo proposto e os softwares envolvidos. Por fim, o capítulo 6 contém as considerações finais e as propostas para trabalhos futuros.

2 REFERENCIAL TEÓRICO

A bioinformática vem se consolidando como uma área de grande importância para a compreensão da função e do comportamento de sistemas biológicos. Ela integra a tecnologia da informação e biologia através da construção de algoritmos, do uso de técnicas computacionais e do desenvolvimento de aplicações web que facilitam sua utilização por usuários das ciências da vida.

2.1 Aplicações Web de Bioinformática

O desenvolvimento de novas ferramentas de software para análise de dados biológicos tem sido um componente-chave para o avanço da área de pesquisa da bioinformática. O crescimento dos dados genômicos impactou o cenário da biologia contemporânea, tornando as ferramentas computacionais uma força primária na pesquisa científica.

Aplicações web de bioinformática têm diferentes finalidades, atendendo a objetivos específicos de cada projeto. Elas podem ser constituídas por diversas tecnologias, arquiteturas distintas e usar diferentes formatos de arquivo para representar os mesmos tipos de dados, além de integrar outros softwares para formar novos pipelines. Nesse contexto, a heterogeneidade desses softwares é evidente. Por exemplo, o sistema de gerenciamento de dados medna-metadata é implementado em Python, utiliza PostgreSQL com PostGIS para dados geográficos, RabbitMQ para mensageria e servidor web NGINX (Kimble *et al.*, 2022). A plataforma GREIN, voltada para a manipulação e análise de sequenciamento de RNA no banco de dados Gene Expression Omnibus, apresenta um pipeline baseado na linguagem R e pacotes R (Mahi *et al.*, 2019). Já o MOLGENIS Research é uma aplicação web para coletar, gerenciar, analisar, visualizar e compartilhar grandes conjuntos de dados biomédicos, implementada em Java 1.8 com suporte do framework Spring MVC, gerenciada pelo Apache Maven, e utilizando servidor web Apache Tomcat, banco de dados PostgreSQL com indexação em ElasticSearch, e interface gráfica com Bootstrap, Vue e templates Freemarker (Velde *et al.*, 2019).

Pesquisadores biológicos e biomédicos têm utilizado essas ferramentas para analisar o grande volume de dados genômicos e, subsequentemente, estabelecer os fundamentos para o desenvolvimento de novas interpretações clínicas. No entanto, apesar das publicações científicas muitas vezes compartilharem dados e código-fonte, não existem garantias a respeito da completude dessa documentação, levando à divulgação de uma documentação defasada e ao desaparecimento do software (Mangul *et al.*, 2019).

Três principais diferenças distanciam aplicações web de bioinformática de softwares produzidos pela indústria:

1. softwares comerciais são desenvolvidos por grandes equipes com desenvolvedores especializados enquanto equipes envolvidas no desenvolvimento de softwares científicos podem não ter formação em engenharia de software. Muitas das aplicações carecem de interfaces focadas na experiência do usuário e um gerenciamento de dependências adequado, tornando a usabilidade e interoperabilidade com outros softwares um processo especialmente complicado para pesquisadores com conhecimento computacional limitado.

2. o método adotado na comunidade de biologia computacional é uma estrutura de curto prazo para disseminar o desenvolvimento de software. Esse método consiste em publicar um artigo descrevendo a ferramenta a fim de demonstrar sua eficácia com conjuntos de dados de amostra. Contudo, os materiais suplementares apesar de disponibilizados junto ao artigo, existem em um local fora do controle direto do meio de publicação. Tanto qualidade quanto disponibilidade estão sujeitos a menos avaliação crítica de revisão.

3. as estruturas acadêmicas oferecem pouca recompensa e os desenvolvedores podem perder apoio mesmo para as ferramentas amplamente utilizadas. A perda desses recursos pode interromper o desenvolvimento impactando potencialmente o avanço científico no campo de atuação da aplicação. Por outro lado, o software industrial recebe incentivos para ser mantido e atualizado enquanto for considerado valioso.

Esses são fatores limitantes da aplicabilidade das ferramentas desenvolvidas e reprodutibilidade dos resultados alcançadas pelas mesmas (Mangul *et al.*, 2019).

As questões apresentadas anteriormente tem se fortalecido à medida que o desenvolvimento das aplicações web passaram de novos algoritmos e simples ferramentas individuais para componentes variados interconectadas em complexos pipelines (Leprevost *et al.*, 2017). O que se observa nesses experimentos divulgados, em sua maioria, é o compartilhamento de sistemas projetados para uso local impulsionando situações de irreprodutibilidade computacional, impactando negativamente sobre o funcionamento de outros softwares devido à carência do gerenciamento de dependências e variações entre ambientes de desenvolvimento, resultando em códigos menos modulares, de difícil manutenção (Bini *et al.*, 2021). Para aproveitar plenamente os dados dos experimentos, a comunidade de biociências precisa adotar tecnologias e mecanismos que promovam a interoperabilidade (Sansone *et al.*, 2012).

2.1.1 Interoperabilidade

Pipelines tradicionais são fortemente acoplados à sua infraestrutura de computação local, carecem de documentação suficiente e versionamento de ferramentas, tornando-os difíceis de compartilhar e manter. Uma alternativa para solucionar esses problemas de

manutenibilidade, portabilidade, reprodutibilidade e interoperabilidade são os chamados gerenciadores de *workflow* que provisionam um *framework* para o funcionamento e monitoramento de pipelines. Considerando o ambiente computacional como parte do experimento *in silico*, a documentação do *workflow* tem particularidades a serem preservadas com vistas à sua reprodutibilidade. Eles oferecem geração de relatórios de execução contendo informações a respeito do ambiente de execução, versão das dependências e parâmetros de cada uma delas, além da visualização das etapas do pipeline. Implementar um pipeline com um desses gerenciadores simplifica o desenvolvimento e manutenção, enquanto permite portabilidade, interoperabilidade e reprodutibilidade (Wratten; Wilm; Göke, 2021).

A reprodutibilidade é fundamental para desenvolver pipelines de análise e pode ser alcançada com pipelines personalizados bem escritos. No entanto, os gerenciadores de *workflow* vão além dos requisitos mínimos, melhorando a proveniência dos dados, a legibilidade e a portabilidade. Isso aumenta a transparência, garante a sustentabilidade a longo prazo dos *workflows* de análise e promove análises computacionais rastreáveis, acessíveis, interoperáveis e reutilizáveis (FAIR, do inglês *findable, accessible, interoperable, and reusable*). A capacidade de desenvolver e compartilhar pipelines colaborativamente é uma das maiores transformações trazidas pelos gerenciadores de *workflow*. Métodos computacionais agora se apoiam em um *workflow*, permitindo que equipes de bioinformática disponibilizem seus pipelines e processem grandes volumes de dados de forma eficiente (Wratten; Wilm; Göke, 2021). Alguns exemplos desses gerenciadores são o Galaxy e o CIPRES Science Gateway.

Fundado em 2005, o Galaxy [1] capacita biólogos sem experiência em programação ou administração de sistemas a realizar análises computacionais pela web. Ele adapta ferramentas de análise existentes para uma interface web, permitindo a execução de análises complexas em várias etapas com a preservação completa da proveniência de cada passo. Ao facilitar a colaboração entre desenvolvedores de ferramentas e pesquisadores, o Galaxy acelera significativamente o progresso científico de ambos os grupos (Afgan *et al.*, 2016).

Originado como parte do projeto CIPRES (*CyberInfrastructure for Phylogenetic REsearch*), o CIPRES Science Gateway [2] proporciona acesso a ferramentas para inferir relações filogenéticas a partir de dados de sequências de DNA e proteínas. Dirigido a pesquisadores e educadores, ele permite que os usuários executem análises sem precisar de conhecimento sobre a complexidade operacional, focado em desenvolver algoritmos e ferramentas para analisar relações filogenéticas em uma vasta gama de organismos. O CSG simplifica o acesso dos pesquisadores a ferramentas da comunidade científica que utilizam recursos computacionais robustos, viabilizando análises evolutivas em conjuntos de dados que seriam complexos de manejar de outra forma (Miller; Pfeiffer; Schwartz, 2011).

[1] https://galaxyproject.org
[2] https://www.phylo.org/index.php

Não obstante, o provisionamento, instalação e configuração de ambientes computacionais para a execução dos workflows requer recursos, tempo e profissionais qualificados (Oliveira, 2019). Além disso, os pipelines de bioinformática variam em complexidade, pois muitos são desenvolvidos para necessidades específicas de projetos e não se adaptam facilmente a ambientes de workflow genéricos. Outra questão é a heterogeneidade dos softwares de bioinformática. Embora disponibilizar pipelines personalizados e complexos por meio de aplicações web facilite o acesso e uso dessas ferramentas pela comunidade científica, essa heterogeneidade pode comprometer a interoperabilidade. O uso de contêineres, como Docker, pode mitigar esse problema ao encapsular todas as dependências de uma aplicação web em um ambiente isolado, facilitando a implantação e garantindo a consistência do ambiente de execução (Silver, 2017).

2.2 Tecnologia de contêineres Docker

O Docker já foi apresentado como uma tecnologia de conteinerização capaz de lidar com os desafios de replicação de software, não só no campo da bioinformática, como também em campos semelhantes de pesquisa, em que a computação e o conhecimento computacional desempenham um importante papel (Boettiger, 2015). Contêineres são uma tecnologia de virtualização leve e configurável que permite o empacotamento e a distribuição de pipelines e suas respectivas dependências de maneira autocontida e independente de plataforma (Silver, 2017), ou seja, um *snapshot* da aplicação. Essa arquitetura reduz significativamente o esforço necessário para portar e manter uma aplicação, contribuindo para interoperabilidade, eficiência e manutenção.

Docker é um software de conteinerização comumente utilizado na bioinformática adotado até mesmo para encapsulamento e compartilhamento de complexos *workflows* para diferentes ambientes. Com o tempo, as dependências são atualizadas e versões de software mudam, resultando em falhas frequentes em scripts que tenham sido pré-configurados. O Docker pode mitigar esses problemas ao fornecer um mecanismo para encapsular ferramentas e suas dependências de maneira altamente portátil (O'Connor *et al.*, 2017).

Contêineres são baseados em imagens, uma imagem de contêiner é um pacote executável que inclui todo o necessário para executar uma aplicação, isso permite ao software funcionar de maneira uniforme, independentemente das diferenças nos sistemas. Em tempo de execução, a imagem se torna um contêiner (Tam *et al.*, 2023). Os autores de *workflows* podem configurar seus ambientes em uma imagem Docker, incluindo ferramentas, dependências, arquivos de referência, e provisionar essa imagem em nuvem ou de forma local. Isso permite a entrega do software e realização de análises em poucos minutos, simplificando estratégias. Contêineres Docker tornam o ambiente de execução consistente e portátil, permitindo o uso de diversos ambientes computacionais (O'Connor *et al.*, 2017).

Projetos como Dockstore (O'Connor *et al.*, 2017) e BioContainers (Leprevost *et al.*, 2017) evidenciam o movimento da comunidade científica em prol do uso de contêineres Docker, destacando a necessidade de migrar aplicações existentes para Docker para garantir compatibilidade com esses e futuros projetos. No entanto, uma revisão sistemática da literatura não identificou processos de engenharia de software no estado da arte que abordem essa finalidade, considerando tanto o contexto único de cada aplicação quanto sua manutenção.

2.3 Ciclo de Vida de Software

O ciclo de vida de software, frequentemente referido como processo de software, "é um conjunto de atividades relacionadas que levam à produção de um produto de software" (Sommerville, 2011, p. 29). Ele constitui a base para o gerenciamento de projetos de software e estabelece o contexto para a produção de artefatos (modelos, documentos, dados, relatórios, formulários, etc.), garantia de qualidade e gestão de mudanças. Dentro desse contexto, um modelo de processo de software é uma representação simplificada de um processo de software. Eles podem ser vistos como frameworks que podem ser ampliados e adaptados para criar processos de engenharia de software mais específicos. Alguns tipos de modelos de processo, de acordo com (Sommerville, 2011), são:

- **Modelo em Cascata:** Este modelo considera as atividades fundamentais do processo de especificação, desenvolvimento, validação e evolução como fases distintas, como especificação de requisitos, projeto de software, implementação, teste, e assim por diante.
- **Desenvolvimento Incremental:** Esta abordagem intercala as atividades de especificação, desenvolvimento e validação. O sistema é desenvolvido em uma série de versões incrementais, onde cada versão adiciona funcionalidade à anterior.
- **Engenharia de Software Orientada a Reúso:** Baseada na existência de um número significativo de componentes reusáveis, esta abordagem foca na integração desses componentes em um sistema existente, em vez de desenvolver um sistema do zero.

Esses modelos podem ser adaptados e utilizados em conjunto para aproveitar as vantagens de cada abordagem (Sommerville, 2011).

Para auxiliar a melhoria contínua desses processos, o SEI (*Software Engineering Institute*) desenvolveu o CMMI (*Capability Maturity Model Integration*). Este framework foi desenvolvido para ajudar organizações a avaliar e aprimorar suas capacidades de planejamento, gestão e qualidade dos produtos entregues. Apesar de reconhecer a importância da

manutenção, o CMMI apresenta lacunas significativas nesse aspecto, como a falta de foco em práticas específicas de manutenção. Por exemplo, a análise de impacto, uma atividade importante na manutenção de software, não é explicitamente abordada na estrutura do CMMI, nem o conceito de maturidade da manutenção. Essas limitações indicam que, embora eficaz na gestão de projetos de desenvolvimento, o CMMI não oferece orientações suficientes para a manutenção de software, uma área onde muitas organizações ainda carecem de processos bem definidos. Apesar disso, o modelo proporciona uma abordagem flexível, permitindo que cada organização selecione os processos a serem melhorados de acordo com suas necessidades específicas (Jansson, 2007).

2.3.1 Processos de Manutenção

O ciclo de vida do software pode ser visto como um processo em sequência iterativa, envolvendo requisitos, projeto, implementação e testes que duram toda a vida útil do sistema (Sommerville, 2011). No entanto, o modelo tradicional de manutenção do software, que pressupõe que uma única organização seja responsável tanto pelo desenvolvimento inicial quanto pela manutenção, não se aplica ao contexto científico. A maioria dos softwares científicos são de código aberto, aceitando contribuições de toda a comunidade e experimentando mudanças frequentes na equipe de pesquisadores.

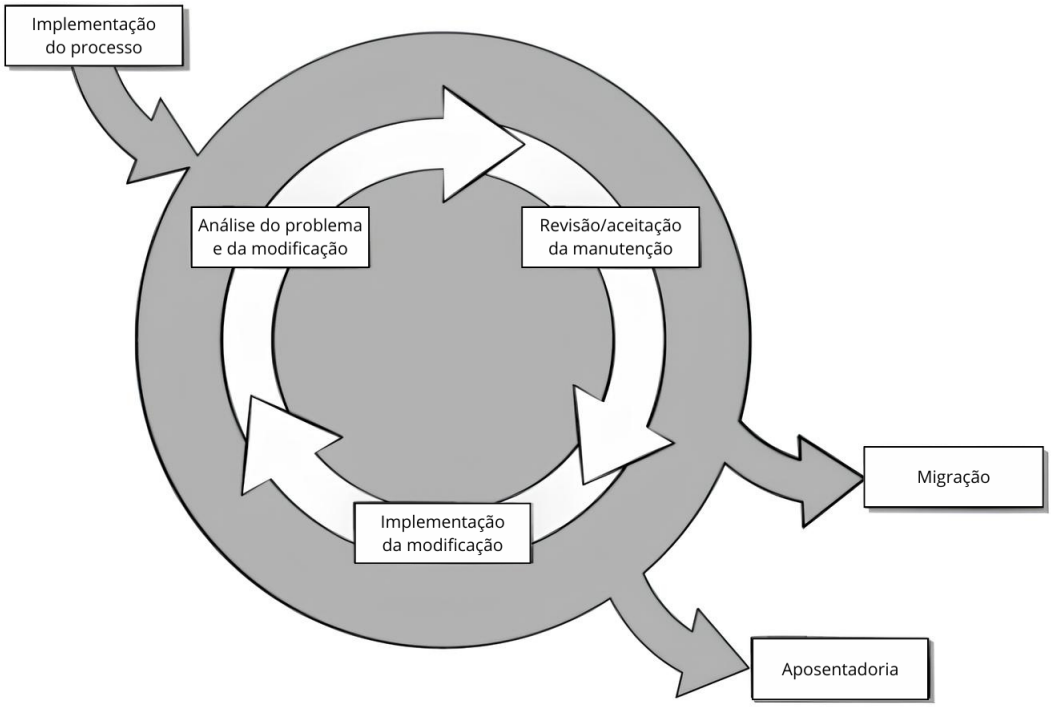
No contexto da bioinformática, a falta de um processo de software robusto pode levar a diversos problemas devido à natureza interdependente dos sistemas. Por exemplo, correções de erros são cruciais em softwares utilizados para análises genômicas, em que resultados imprecisos podem ter sérias implicações. Além disso, a adaptação a novas tecnologias e plataformas é essencial em um campo que está em constante evolução, com novas técnicas e instrumentos sendo desenvolvidos regularmente. A dependência de outros sistemas é comum na bioinformática, onde softwares muitas vezes precisam integrar dados de diferentes fontes e interagir com diversas ferramentas de análise, essa complexidade retrata uma situação caracterizada por (Hopkins; Jenkins, 2008) como "desenvolvimento de software *brownfield*", na qual os sistemas precisam evoluir em um ambiente em que dependem de outros sistemas de software.

A necessidade de evoluir em um contexto de interoperabilidade aumenta as dificuldades e os custos de manutenção. Além de entender e analisar o impacto de uma mudança proposta no próprio sistema, é importante avaliar como isso pode afetar outros sistemas no ambiente (Sommerville, 2011).

A importância de um processo de manutenção de software, muitas vezes também chamado de processo de evolução, não pode ser subestimada, especialmente em áreas como a bioinformática. A falta de um processo de manutenção claro pode levar a falhas na integração, erros nos resultados das análises e aumento nos custos operacionais. Na figura 1 é mostrado um modelo de processo de manutenção em que são considerados também os

cenários de migração para uma versão mais nova do software e aposentadoria, em que se trabalha no desenvolvimento de um substituto.

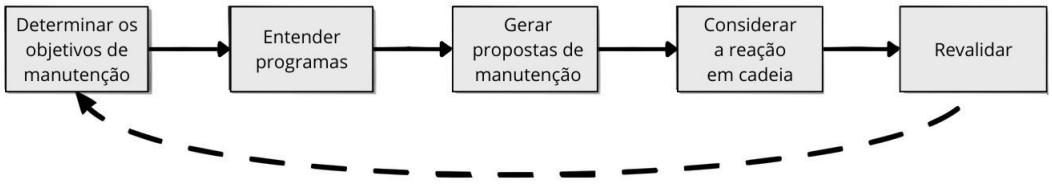
Figura 1 – Processo Genérico de Manutenção de Software



Fonte: (Pressman; Maxim, 2021)

A figura 2 mostra um conjunto de tarefas genéricas que devem ser completadas como parte de um processo controlado de manutenção de software. A definição de um processo de manutenção que considere a interoperabilidade e a evolução constante dos softwares é vital. Manter aplicações atualizadas, corrigir defeitos, adaptar a novas tecnologias e melhorar o desempenho são tarefas essenciais para garantir a longevidade e a eficácia das aplicações.

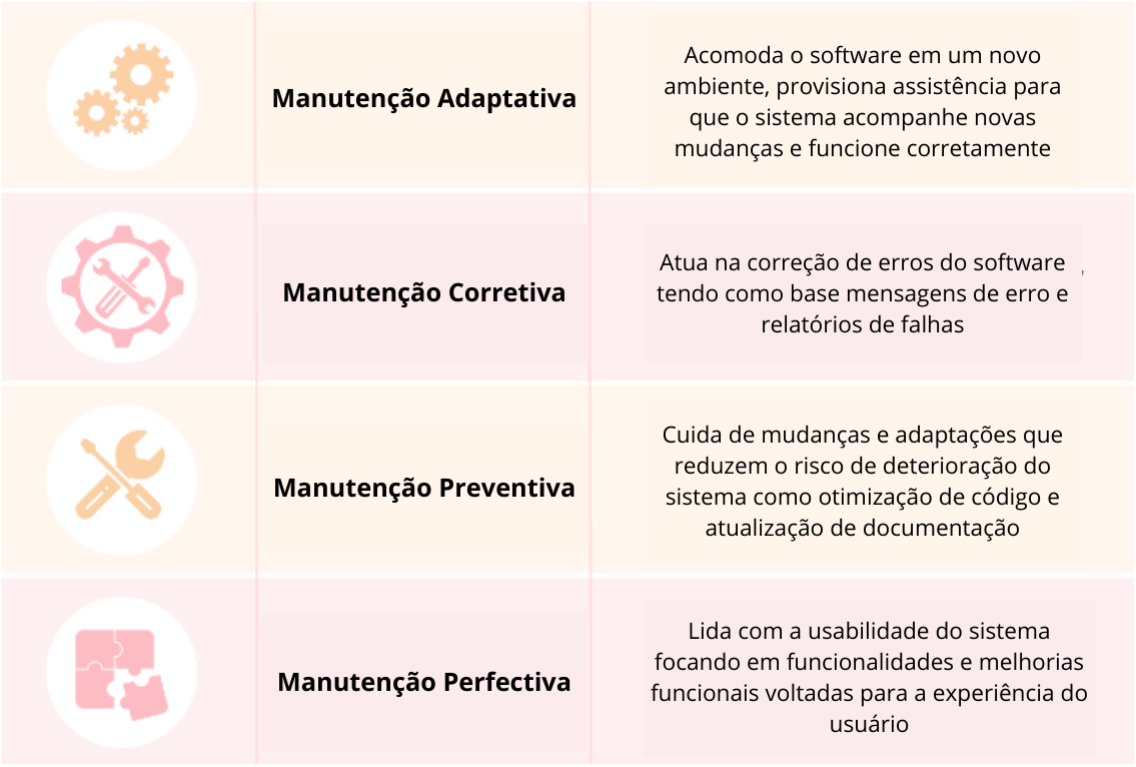
Figura 2 – Tarefas de Manutenção de Software



Fonte: (Pressman; Maxim, 2021)

Os tipos de manutenção de software apresentados por (Pressman; Maxim, 2021), estão exibidos na figura 3. A evolução das aplicações para lidar com a adaptação de novos ambientes e requisitos são as atividades que consomem mais esforço de manutenção, sendo

Figura 3 – Tipos de Manutenção de Software



Fonte: Adaptado de (Pressman; Maxim, 2021)

que a manutenção, por si só, chega a deter dois terços dos recursos destinados ao ciclo de vida do sistema (Sommerville, 2011). "A tarefa de criar artefatos de software que facilitam o suporte e a manutenção exige uma engenharia cuidadosa e consciente" (Pressman; Maxim, 2021, p. 175), portanto, um processo bem estruturado para essas atividades é determinante para assegurar que as aplicações de bioinformática tenham longevidade no ciclo de vida e evitem custos onerosos aos recursos de pesquisa da área.

2.4 Trabalhos Correlatos

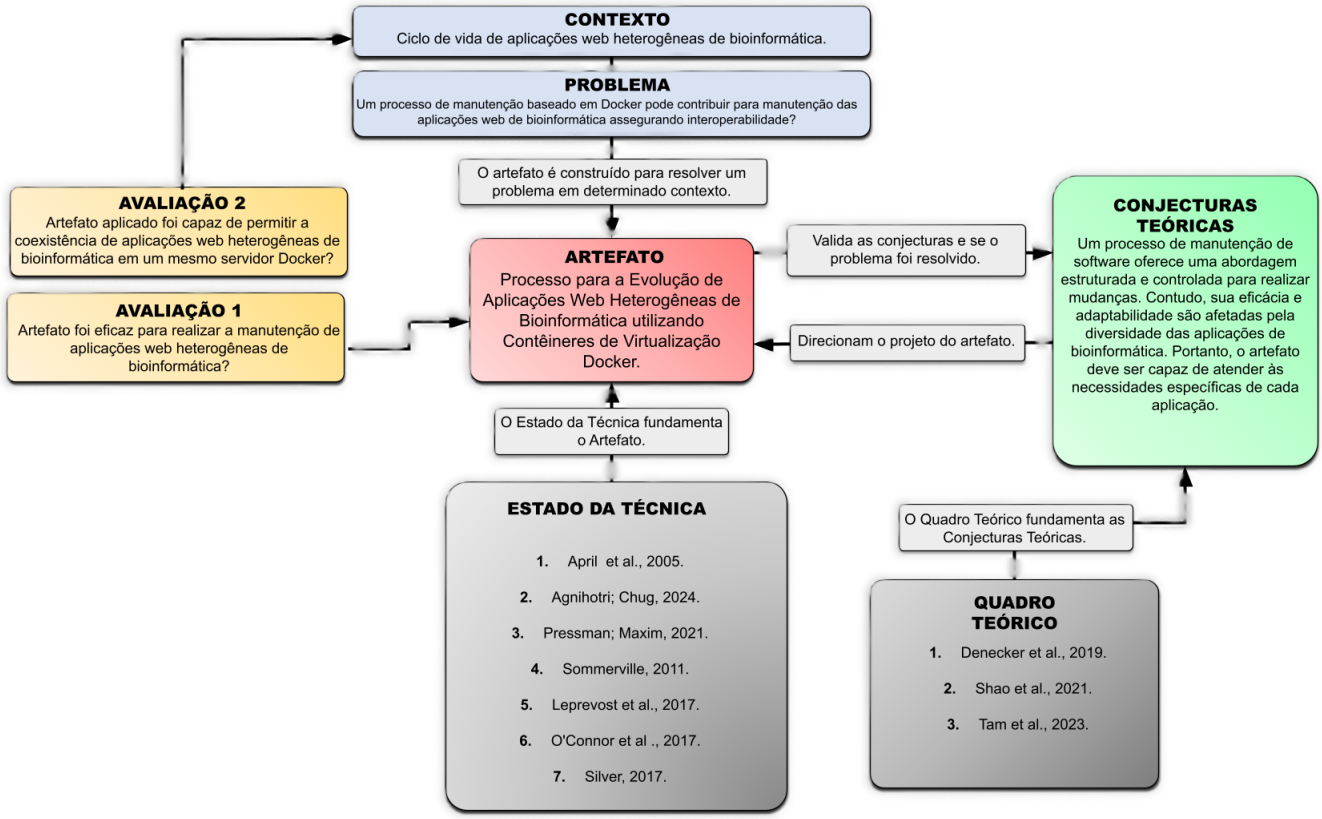
Através da revisão de literatura, foi possível observar que a estratégia de conteinerização em Docker já vem sendo utilizada em aplicações de bioinformática com diferentes graus de complexidade onde destacam-se dois cenários. No primeiro cenário estão as aplicações que possuem interação entre diversos serviços, como banco de dados, gerenciador de filas, serviço de mensageria e servidor web, onde cada serviço poderia ser isolado em um contêiner distinto como visto em (Shao *et al.*, 2021) e (Denecker *et al.*, 2019). Além disso, foram observadas interações entre aplicações como implementações de algoritmos de busca em banco de dados de bioinformática ou implementações de pacotes na linguagem R, observados respectivamente em (Degnan *et al.*, 2021) e (Mahi *et al.*, 2019). Por outro lado, no segundo cenário estão as aplicações em que todo o ambiente necessário é replicado em um único contêiner, como as apresentadas em (Carlucci *et al.*, 2019) e

(Uriarte; Herrera-Nieto, 2022). Esse contexto é comumente observado em aplicações que possuem sua implementação baseada em linguagem R. Contudo, para ambos os cenários, não foi encontrada no estado da arte uma fundamentação em processo de software e a escolha da abordagem de conteinerização parece ter ocorrido de forma arbitrária. A tabela 3 em anexo demonstra as tecnologias apresentadas nos trabalhos encontrados na revisão de literatura e uma avaliação de relevância baseada na presença de uma seção no artigo que descrevesse o processo de conteinerização.

3 METODOLOGIA DE PESQUISA

A metodologia adotada neste trabalho é a abordagem de pesquisa *Design Science Research* (DSR) (Goecks *et al.*, 2021), baseada no desenvolvimento e avaliação de artefatos para resolução de problemas. Essa metodologia combina aspectos de pesquisa científica e desenvolvimento prático, buscando produzir conhecimento aplicado e soluções tangíveis. A figura 4 mostra uma adaptação da DSR para este projeto, baseado em (Pimentel; Filippo; Santos, 2020).

Figura 4 – Adaptação da *Design Science Research* para este projeto



Fonte: O autor

Quanto à natureza, trata-se de uma pesquisa aplicada, pois visa à elaboração de um processo de software baseado em Docker para a área de bioinformática. O objetivo principal é desenvolver um artefato que possa contribuir para o ciclo de vida de aplicações web de bioinformática, independente do grau de complexidade da aplicação.

Os procedimentos adotados nesta pesquisa seguem as etapas do DSR, que incluem a identificação clara do problema, definição dos objetivos do artefato, concepção e implementação do processo de software, a avaliação da eficácia e a comunicação dos resultados obtidos.

No contexto deste projeto o DSR pode ter suas etapas relacionadas da seguinte maneira:

- **Identificação do problema:** Um processo de manutenção baseado em Docker pode contribuir para manutenção das aplicações web de bioinformática assegurando interoperabilidade?
- **Definição dos objetivos do artefato:** Direcionar a manutenção de aplicações web de bioinformática para que aspectos como interoperabilidade, qualidade e reprodutibilidade sejam atendidos satisfatoriamente.
- **Desenvolvimento do artefato:** Projetar e desenvolver o processo de software baseado em Docker, tendo em vista a diversidade das aplicações web de bioinformática. Isso envolverá a seleção e adaptação de processos voltados para evolução e manutenção de softwares identificando as etapas e atividades necessárias no contexto do G2BC. A eficácia e a adaptabilidade do processo de manutenção de software são afetadas pela diversidade das aplicações web de bioinformática. Para lidar com essa diversidade, pode ser necessário ajustar as etapas do processo e introduzir etapas condicionais para trabalhar com diferentes níveis de complexidade e heterogeneidade das aplicações, alterações que ocorrerão a partir do caráter iterativo do DSR. Isso implica em personalizar o processo de manutenção para atender às necessidades específicas de cada aplicação, garantindo assim uma evolução eficaz e controlada do software.
- **Avaliação do artefato:** Configurar aplicações web de bioinformática em um mesmo servidor para validar a eficácia do processo. Essa etapa será refletida em duas avaliações. A avaliação 1 consiste em dois ciclos de manutenção com escala crescente de complexidade em aplicações web mais um ciclo onde o processo será utilizado e avaliado por usuários de uma equipe de manutenção que irão fornecer dados para uma avaliação qualitativa, totalizando três ciclos. De forma complementar, a avaliação 2 tem o intuito de avaliar a eficácia do processo. No capítulo 5 são apresentados mais detalhes sobre essa avaliação, bem como a métrica utilizada.
- **Refinamento do artefato:** Refinar o artefato para melhorar sua eficácia e adaptabilidade. A validação do processo ocorrerá em ciclos, permitindo seu refinamento, à medida que os dois primeiros ciclos compreenderão aplicações web heterogêneas de complexidade crescente, que passarão pelo processo de manutenção, e o terceiro e último ciclo terá o processo avaliado de forma qualitativa com base na experiência de um grupo de usuários ao utilizar o processo.
- **Comunicação dos resultados:** Documentar, relatar em monografia e apresentar os resultados do projeto, descrevendo o artefato desenvolvido, os refinamentos aplicados e os *insights* obtidos durante o processo.

4 BIODOCKFLOW

Em alinhamento com os objetivos deste projeto e aproveitando a natureza iterativa da metodologia de pesquisa adotada, o DSR, foram definidos ciclos de desenvolvimento que permitiram o refinamento e a complementação do processo. Isso abrange não apenas a manutenção adaptativa, mas também os demais tipos de manutenção, detalhando todas as atividades, ferramentas e componentes envolvidos. Assim, optou-se por criar um processo que evolui conforme as necessidades das aplicações, com suas respectivas atividades sendo aprimoradas com base em trabalhos correlatos, na documentação oficial do Docker e na experiência prática adquirida.

Foram conduzidos três ciclos de definição do processo, os dois primeiros se diferenciam por um maior grau de complexidade das aplicações submetidas às fases, enquanto o último teve como foco analisar a experiência de integrantes de uma equipe de manutenção ao utilizar o processo. Além disso, cada ciclo tinha objetivos, alinhados aos objetivos específicos do projeto. Nas seções seguintes, cada ciclo é detalhado. As versões do BioDockFlow, incluindo a atual, estão disponíveis em seu repositório do GitHub[1], contudo a versão atual resultante do último ciclo também encontra-se nos anexos B e C.

4.1 Ciclo 1

Este primeiro ciclo teve como objetivo definir um rascunho de processo com atividades que permitam a manutenção adaptativa de conteinerização Docker de aplicações com grau de complexidade baixo. Foram identificadas atividades com base em experiência prática pessoal e observando-se a principal característica das aplicações web de bioinformática às quais este processo se aplica: sua heterogeneidade decorrente do acoplamento de dependências. As atividades foram as seguintes:

1. **Analisar do gerenciamento de dependências da tecnologia de implementação:** diferentes tecnologias usam diferentes formas de lidar com suas dependências, algumas possuem gerenciadores que conseguem resolver com apenas um comando, outras, além da instalação, necessitam que os pacotes adicionais sejam habilitados em um arquivo de configuração e é possível até mesmo que todo o necessário esteja em um diretório de pastas da própria aplicação.

2. **Analisar dos serviços que precisam interagir de forma independente:** diferentes serviços como banco de dados precisam ser devidamente acoplados e isolados em seus próprios contêineres em prol de garantir as especificações de versões das aplicações.

[1] https://github.com/G2BC/BioDockFlow

3. **Analisar dos volumes que precisam ser criados:** a persistência dos banco de dados em contêineres é mantida por meio de volumes Docker, da mesma forma, diretórios que guardam as dependências da aplicação devem ser referenciados para o contêiner como um volume.

4. **Definir cenário:** de acordo com os cenários 1 e 2 identificados na revisão sistemática.

5. **Implementar Dockerfile:** a análise da tecnologia de implementação mostrará como a mesma lida com o gerenciamento de dependências e havendo a existência do gerenciador e a necessidade de instalação das dependências um arquivo Dockerfile deve ser criado, apontando como base a versão da imagem oficial da tecnologia que seja necessária, e realizando a instalação das dependências.

6. **Implementar docker-compose.yml:** a análise dos serviços e dos volumes indicará como o docker-compose.yml estará estruturado, havendo a existência dos serviços, este arquivo irá gerenciar os múltiplos contêineres e, da mesma forma, os volumes que serão atribuídos a cada um deles.

7. **Realizar *build* do contêiner e executar:** realizadas as atividades posteriores, o contêiner pode ser construído utilizando os arquivos feitos e a conteinerização da aplicação estará pronta para ser testada.

8. **Realizar testes de funcionalidade:** realizar teste exploratório da implementação para garantir o funcionamento e interação corretos entre os serviços isolados em cada contêiner.

9. **Configurar rede Docker:** configuração de uma rede Docker para que os contêineres possam se reconhecer através do seus nomes de host, evitando preocupações com os endereços de IP atribuídos a cada um.

10. **Configurar proxy:** configuração de acesso a aplicação por meio de um proxy, também implementado em contêiner devido a natureza do processo, reforçando a necessidade da configuração da rede Docker.

O rascunho inicial proveniente dessas atividades promoveu a base do artefato para submissão das primeiras aplicações ao processo de manutenção, permitindo a validação prática, detalhada na seção 5.1.1, aliada a identificação de melhorias que irão compor o processo no ciclo seguinte. A versão 1 do BioDockFlow resultante deste ciclo encontra-se no repositório do GitHub [2].

[2] https://github.com/G2BC/BioDockFlow/tree/main/v1

4.2 Ciclo 2

A definição de um processo de software compreende a organização de fases, fluxo de trabalho, atividades e ferramentas de apoio. Neste ciclo objetivou-se evoluir o processo para atender aplicações de complexidade média, com arquiteturas de serviços *frontend* e *backend*, e incluir no processo todos os tipos de manutenção encontrados na literatura, com seus respectivos fluxos de trabalho, contemplando assim o segundo objetivo específico deste projeto.

4.2.1 Fases do processo

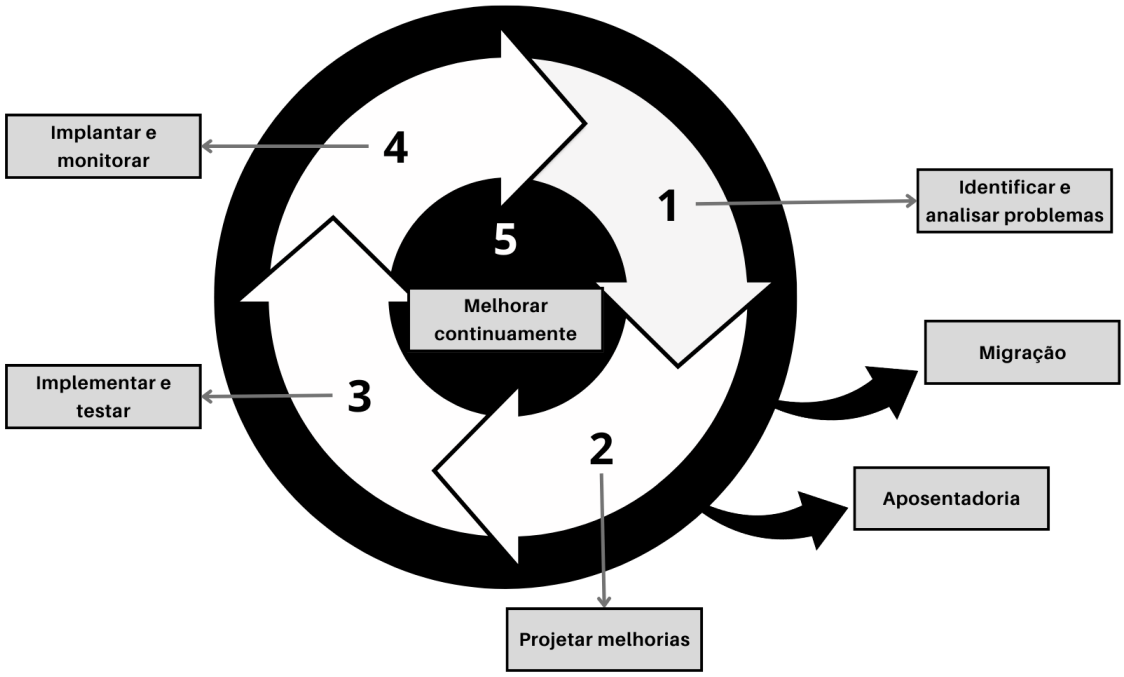
A partir da análise da literatura sobre manutenção de software, cumprindo com o primeiro objetivo específico, foram propostas fases para o processo de manutenção. O processo se baseia principalmente no modelo proposto por (Pressman; Maxim, 2021), incluindo algumas adaptações. Em um nível macro, as fases do processo podem ser identificadas como: 1) Identificar e analisar problemas; 2) Projetar melhorias; 3) Implementar e testar; 4) Implantar e monitorar; e 5) Melhorar continuamente.

A fase um está no escopo de identificação e categorização dos problemas; as fases dois e três envolvem planejamento e implementação distintos para cada categoria; a fase quatro contempla o *deploy* para produção; a fase cinco envolve revisar e atualizar regularmente o software e o próprio processo para manter o desempenho ideal. A figura 5 ilustra a prioridade dessas fases e como devem ser vistas no contexto de interação, a quinta fase é posicionada ao centro evidenciando o seu caráter flexível, a otimização pode ocorrer durante e sobre qualquer uma das outras fases. Ao longo dessa seção são detalhadas cada uma dessas fases.

A fase 1, identificar e analisar problemas, é acionada por diversas fontes, como a análise do código-fonte, a revisão da documentação, a identificação de defeitos provenientes de testes ou validações em homologação, solicitações de mudanças pelos usuários e mudanças emergentes devido a alterações no ambiente de produção. As questões levantadas serão categorizadas: problemas identificados em testes e validações podem ser classificados como correção de bugs, novas solicitações de usuários como solicitação de mudanças, a análise do código-fonte pode indicar a necessidade de refatoração, a revisão da documentação pode apontar a necessidade de atualização, e questões relacionadas ao ambiente de produção podem sugerir a necessidade de conteinerização com Docker para tornar o software mais independente. No entanto, a definição da categoria só poderá ser feita com precisão após uma análise detalhada, que identificará o risco e o impacto em outros módulos do software.

A segunda fase, projetar melhorias, envolve o planejamento da solução com base na categoria definida na fase anterior. Cada categoria possui atividades de análise e desenvolvimento específicas, porém, o foco principal aqui é identificar o que será feito e

Figura 5 – Fases do processo de manutenção



Fonte: Adaptado de (Pressman; Maxim, 2021)

como será realizado. Nessa fase, são realizadas análises das condições em que a manutenção ocorrerá, bem como a avaliação de logs e mensagens que possam auxiliar na solução dos problemas. Dependendo das necessidades, a equipe de manutenção pode optar por trabalhar em QMs de forma isolada ou abordar um conjunto de QMs simultaneamente, projetando-as juntas e desenvolvendo-as em paralelo, dentro de um mesmo período, para que esse conjunto forme uma entrega única.

Após a fase de projetar melhorias, a terceira fase envolve a implementação da solução, seguindo o escopo definido nas fases anteriores. Neste estágio, a solução é desenvolvida, testada e integrada ao software conforme o planejamento estabelecido. É fundamental que cada responsável por uma QM realize testes de funcionalidade ao final de sua atividade, a fim de garantir que a implementação foi correta, ou seja, que não gerou novos pontos de manutenção em potencial e não impactou negativamente outros módulos do sistema.

A quarta fase engloba as atividades de *deploy*, como a atualização da versão do software em seu repositório, a preparação do ambiente de produção e a disponibilização do sistema para os usuários. No contexto do G2BC, devido à infraestrutura oferecida pelo servidor da UNEB, essa fase envolve uma colaboração entre a equipe de manutenção e uma equipe especializada em infraestrutura e redes. Juntas, elas realizam a solicitação de URLs, a configuração de certificados digitais e a configuração do *proxy* no servidor para tornar as aplicações acessíveis através das URLs fornecidas. Além disso, o software deve ser monitorado regularmente para garantir a otimização de desempenho e seu pleno funcionamento no ambiente de produção.

A melhoria contínua representa a última fase do processo e envolve a revisão regular e a atualização do sistema, com o objetivo de manter a otimização de desempenho. Essa fase é aplicada tanto ao software quanto ao processo em si, considerando que estratégias específicas de manutenção podem ser desenvolvidas, revisadas e ajustadas conforme as necessidades ao final de cada ciclo de manutenção. Caso um conjunto de QMs seja abordado, ao término do período estipulado para a implementação de todas as atividades, será realizada uma reunião com a equipe de manutenção para identificar os pontos de melhoria e adaptar o processo ao contexto da equipe.

A fase de migração envolve a transição de um software existente para uma nova versão, seja por razões tecnológicas, de desempenho ou por necessidade de modernização. Pressman & Maxim (2021) destaca que, durante essa fase, podem estar envolvidas atividades de reengenharia, como refatoração de código, refatoração de dados e engenharia reversa, e até mesmo a adoção de um modelo de processo de reengenharia.

Por fim, a fase de aposentadoria ocorre quando o software se torna obsoleto e não se tem uma resposta positiva da análise de custo-benefício para que seja realizada a manutenção. Segundo Pressman & Maxim (2021), o desenvolvedor ou equipe de manutenção deve se planejar para começar a criar um substituto, seguindo práticas mais modernas de engenharia, antes que os clientes abandonem o sistema. Dessa forma a equipe caminha para o desenvolvimento de uma aplicação de alta qualidade.

4.2.2 Fluxo de trabalho e atividades do processo

Visando atender não apenas aos diferentes tipos de manutenção presentes na literatura, além da manutenção adaptativa, mas também às categorias de documentação, refatoração, solicitação de mudanças, correção de bugs e Docker, foi elaborado um fluxo de trabalho composto pelas fases descritas na seção 4.2.1. Para cada fase, foram definidas atividades que abrangem os quatro tipos de manutenção propostos por Pressman & Maxim (2021). Essas atividades seguem um fluxo de trabalho específico, orientado pela categorização da QM, respeitando as particularidades de cada categoria.

Cada atividade é composta por:

- **Fase:** indica a fase do processo a qual a atividade faz parte;
- **Atividade:** definição da atividade de forma clara com um verbo no infinitivo;
- **Descrição:** detalhamento geral da atividade a respeito do que deve ser feito e, em alguns casos, como deve ser feito;
- **Pré-condições:** aponta o estado requerido pelo sistema para que seja cumprida a atividade, comumente aplicadas às atividades que dependem do resultado de outras e alteram o estado do software;

- **Entradas:** artefatos necessários para realização das tarefas;
- **Tarefas:** itens que devem ser cumpridos como um *checklist* para a realização da atividade de forma íntegra. Contudo, algumas atividades não possuem um padrão por depender da estrutura e contexto de cada sistema, dessa forma, estas não possuem tarefas especificadas;
- **Ferramenta:** indica a ferramenta na qual deve ser realizada a atividade;
- **Saídas:** artefatos resultantes da atividade; podem ser artefatos novos ou os mesmos artefatos da entrada, caso envolvam modificações dos mesmos;
- **Observações:** informações adicionais para realização da atividade, como padrões a serem seguidos;

Foram criadas 18 atividades distribuídas entre as 4 principais fases do processo. Na fase 1, identificar e analisar problemas, foram criadas duas atividades: 1. Registrar QM; 2. Analisar QM.

A partir da categorização das questões de mudança na fase 1 do processo, o fluxo de trabalho é subdividido em três subfluxos: um para Docker, composto pelas atividades de containerização definidas no ciclo 1, outro para documentação e o terceiro para as categorias de refatoração, correção de bug e mudança de requisitos. As atividades a seguir foram criadas para o segundo subfluxo: 1. Identificar funcionalidades da aplicação; 2. Identificar funcionalidades ausentes na documentação; 3. Identificar instruções de usabilidade ausentes na documentação; 4. Atualizar documentação; 5. Validar documentação; e 6. Atualizar documentação em produção.

A categoria de correção de bug possui o diferencial de ter suas QMs engatilhadas através de mensagens de erro. As mesmas podem ser analisadas para se identificar a origem, bem como os logs da aplicação para entender o contexto que deu origem ao *bug*. As demais atividades, apresentadas a seguir, compõem o subfluxo unificado com as outras duas categorias, sendo assim, para o terceiro subfluxo temos: 1. Análise de mensagens de erro, logs da aplicação; 2. Identificar módulos afetados; 3. Identificar nível de dependência entre os módulos; 4. Priorizar ordem de refatoração dos módulos com base no nível de dependência; 5. Iniciar *branch*; 6. Desenvolver solução; 7. Realizar testes; 8. Atualizar aplicação em produção; 9. Publicar alterações; e 10. Monitorar.

4.2.3 Ferramenta de apoio ao processo

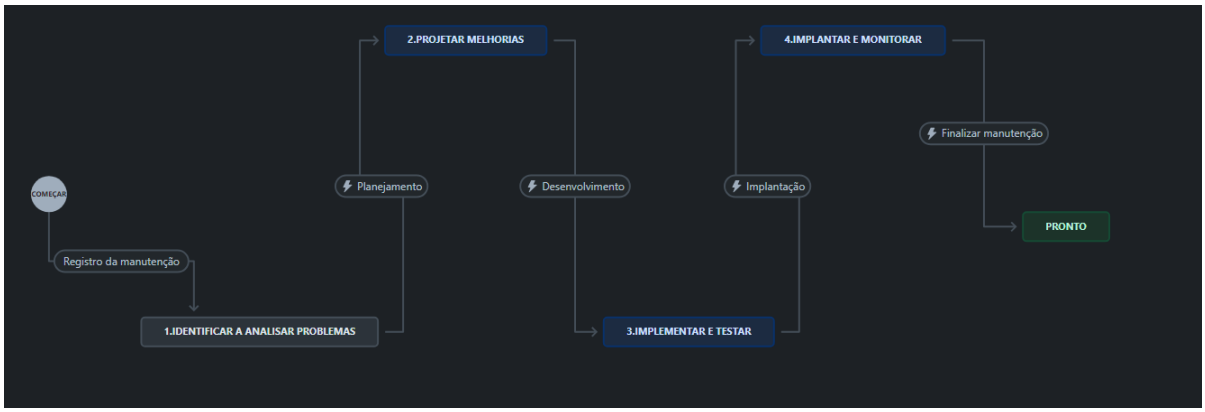
Conforme Sommerville (2011), os processos são apoiados por ferramentas de acordo com as necessidades dos desenvolvedores, por exemplo, ferramentas como testes automatizados e gerenciamento de configuração podem apoiar processos de desenvolvimento. No

caso do BioDockFlow, é fundamental adotar uma ferramenta que dê suporte a todas as fases e atividades necessárias para solucionar as QMs. Uma ferramenta facilita o registro das QMs identificadas e armazena as realizadas, auxiliando no gerenciamento de mudanças e na rastreabilidade, simplificando a gestão e monitoramento das atividades realizadas. As ferramentas sustentam a aplicação dos métodos.

Para sanar essas necessidades, foi escolhido o Jira [3] para dar apoio a execução do processo. Essa ferramenta permite o monitoramento das QMs e acompanhamento de projetos garantindo o gerenciamento das atividades ao longo das fases, possuindo customização facilitada nos atributos de cadastro das QMs. Ferramentas dessa natureza elucidam os pontos de melhoria do processo, facilitam a identificação dos fatores que deram origem a esses problemas e em qual contexto de manutenção eles surgiram. Portanto, o Jira se adequa ao contexto iterativo, às necessidades de registro das QMs e auxilia diretamente no refinamento do processo.

A ferramenta foi customizada para apoiar a execução do BioDockFlow, adequando seu fluxo de trabalho às fases específicas do processo, como mostrado na figura 6. Após a configuração, as fases foram mapeadas em um quadro visual, permitindo o monitoramento claro e contínuo das QMs como pode ser visto na figura 7.

Figura 6 – Fluxo de trabalho BioDockFlow no Jira

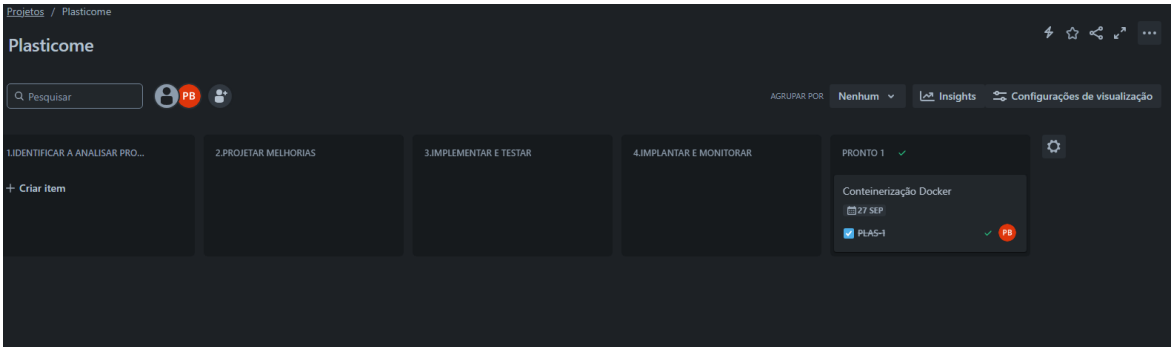


Fonte: O autor

Além disso, a ferramenta permite personalizar os atributos das QMs separando-os em campos de descrição, que descrevem a QM e são exibidos em destaque, e campos de contexto que dão contexto ao trabalho, ajudam a agrupar, filtrar e relatar itens semelhantes. A figura 8 demonstra a seção de customização dos campos e os campos que foram configurados para as QMs deste processo. Já a figura 9 demonstra uma QM, que passou pelo processo, e seus atributos. As transições entre fases podem ser validadas automaticamente, garantindo que uma QM só avance quando todos os critérios forem cumpridos, evitando inconsistências no fluxo de trabalho. A figura 10 exibe a validação criada para a transição da primeira fase para a segunda.

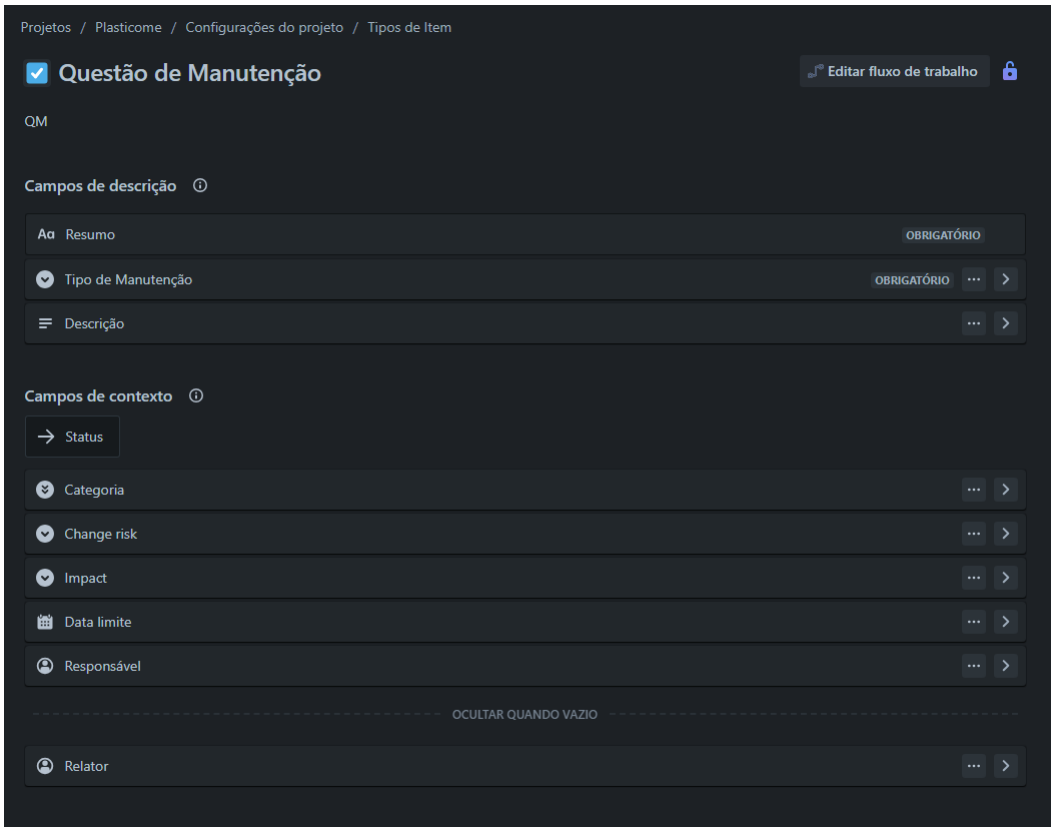
[3] https://www.atlassian.com/br/software/jira

Figura 7 – Quadro de fases no Jira



Fonte: O autor

Figura 8 – Customização dos atributos das QMs

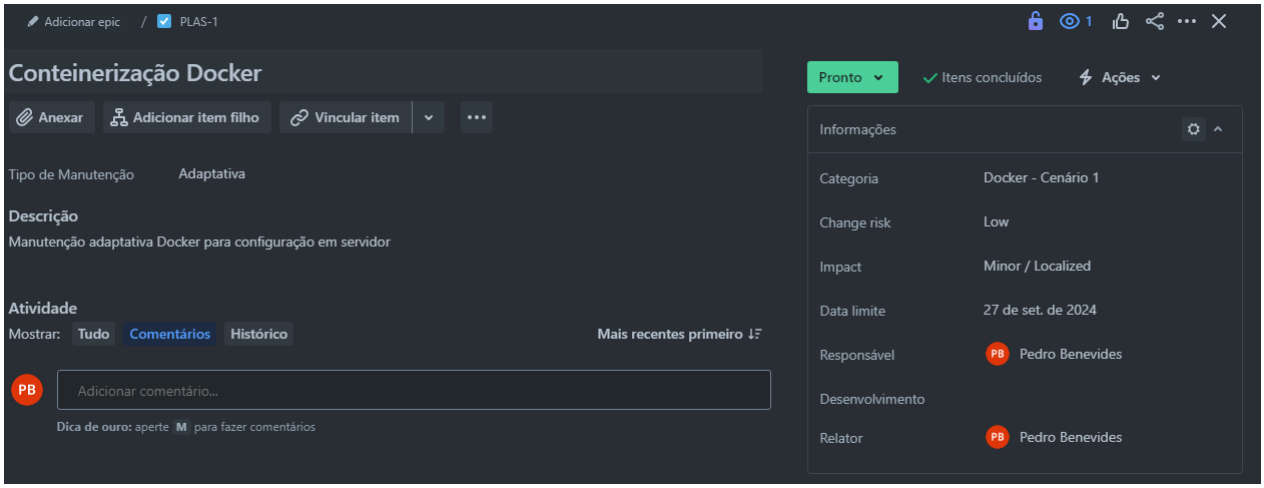


Fonte: O autor

O Jira também facilita o registro de subtarefas dentro de um item principal, permitindo que cada aspecto de uma QM seja acompanhado individualmente, como pode ser visto na figura 11. Isso fortalece a capacidade de dar suporte para execução do processos e das atividades em cada fase e mantém o histórico de todas as ações tomadas contribuindo para um gerenciamento mais eficiente das QMs.

O ciclo 2 provisionou robustez ao processo por meio da inclusão dos tipos de manutenção descritos na literatura, da definição do fluxo de trabalho e da ferramenta de apoio. A avaliação desse ciclo é apresentada na seção 5.1.2, onde são detalhados os resultados relacionados à resolução da QM de conteinerização do PLASTICOME, às

Figura 9 – QM registrada que passou pelo processo



Fonte: O autor

demais QMs identificadas e, por fim, à definição e ao cálculo da métrica utilizada na avaliação quantitativa. Um maior detalhamento das atividades e o fluxo de trabalho da 2ª versão do BioDockFlow estão documentados e disponíveis através do seu repositório no GitHub[4].

4.3 Ciclo 3

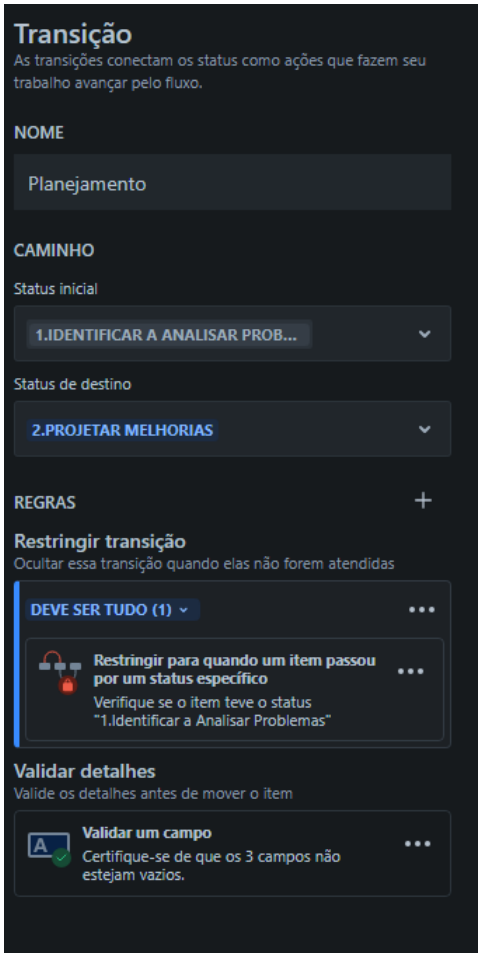
De acordo com o terceiro objetivo específico do projeto, o objetivo do ciclo 3 foi realizar uma avaliação qualitativa baseada na experiência de usuários com o processo. A 2ª versão do BioDockFlow resultante do ciclo 2 foi utilizada por uma equipe de manutenção do grupo G2BC a fim de refinar o processo em um contexto real de manutenção de aplicação web.

Como resultado, identificou-se a necessidade de separar a categoria "solicitação de mudanças" em duas novas categorias: "novos requisitos" e "mudança de requisitos", com o objetivo de aprimorar a caracterização das QMs cadastradas. Nesse contexto, foram incorporadas novas atividades na etapa 3) Implementar e testar, incluindo a abertura de *pull requests*, a realização de *code reviews* e a atualização da aplicação em desenvolvimento. No entanto, considerando que essas atividades estão relacionadas à dinâmica da equipe de manutenção, a qual pode variar em função do número de integrantes, tais atividades passam pela avaliação da condição de adoção de *code review*, ou seja, revisão de código pelos outros integrantes da equipe, garantindo a flexibilidade e adaptabilidade do processo ao contexto específico da equipe.

A seção 5.1.3 apresenta a avaliação deste ciclo, incluindo o sistema submetido ao processo, a quantidade de QMs registradas na ferramenta e sua distribuição por categoria,

[4] https://github.com/G2BC/BioDockFlow/tree/main/v2

Figura 10 – Customização de regras de transição



Fonte: O autor

Figura 11 – QM registrada com atividade

Fonte: O autor

os resultados da resolução da QM de conteinerização, o cálculo da TR-QM, bem como uma análise das respostas dos usuários após a utilização do processo. O fluxo de trabalho resultante do ciclo 3 e as atividades adicionadas estão documentadas respectivamente nos anexos B e C e no repositório do GitHub[5].

[5] https://github.com/G2BC/BioDockFlow

5 AVALIAÇÃO DO ARTEFATO

Conforme proposto na metodologia, a avaliação do artefato foi conduzida em duas etapas: Avaliação 1 e Avaliação 2.

A Avaliação 1, tem o intuito de avaliar a eficácia (quantitativa e qualitativa) do processo na manutenção de aplicações web de bioinformática e foi realizada em ciclos permitindo o refinamento do artefato. Seguindo o caráter iterativo, o ciclo 1 não possui aplicação de métricas de eficácia por conta da maturidade inicial do processo, o ciclo 2 conta com a métrica de avaliação quantitativa e, por fim, o ciclo 3 apresenta ambas as formas de avaliações.

De forma complementar, Avaliação 2 consiste na avaliação qualitativa da interoperabilidade das aplicações web no servidor Docker, ou seja, o funcionamento conjunto das aplicações sem que uma afete o funcionamento da outra.

5.1 Avaliação 1

Composta por um total de três ciclos sendo dois ciclos iterativos, com complexidade e heterogeneidade crescentes das aplicações mais um ciclo de utilização do processo por um grupo de usuários que forneceram dados para uma avaliação qualitativa, considerando o contexto do G2BC. O código-fonte de cada aplicação está disponível no repositório: https://github.com/G2BC.

5.1.1 Ciclo 1

O primeiro ciclo de implementação do processo abrangeu aplicações com baixo grau de dependência tecnológica, o site do G2BC e a aplicação IVET, e as atividades, inicialmente definidas, foram suficientes para a etapa de conteinerização dessas aplicações. No entanto, ao passar por este ciclo, foi possível identificar melhorias e adicionar atividades para a proxima versão, visando atender aos requisitos do ambiente de produção.

- **IVET (*Influenza Viral Evolution Tool*):** A aplicação IVET [1] foi desenvolvida com o objetivo de comparar a evolução de linhagens virais, identificando potenciais linhagens pandêmicas. Esta ferramenta fornece uma interface intuitiva para análise genômica e visualização de dados, facilitando a identificação de mutações e padrões evolutivos em vírus influenza. Baseada em linguagem PHP; necessita manutenção corretiva e adaptativa com foco em conteinerização e configuração em servidor; possui

[1] https://github.com/G2BC/ivet

baixo grau de dependências, portanto, baixa complexidade. Tela do IVET pode ser vista na figura 12.

- **Site G2BC:** O site do G2BC [2] é uma aplicação *Wordpress* baseada em PHP com banco de dados MySQL; necessita de manutenção adaptativa com foco em migração de servidor. Aplicações provenientes da estrutura *wordpress* possuem baixo grau de dependências, as mesmas podem ser facilmente resolvidas ao replicar a estrutura de pastas com todo o conteúdo do site, portanto, baixa complexidade. Tela do site do G2BC pode ser vista na figura 13.

Figura 12 – Tela da aplicação IVET

Fonte: O autor

Figura 13 – Tela inicial do site G2BC

Fonte: O autor

Na aplicação IVET, a análise do gerenciamento de dependências da tecnologia de implementação resultou na seleção da imagem base php:8.1-apache. Esta imagem fornece o

[2] https://github.com/G2BC/g2bc-web

suporte necessário para a versão do PHP utilizada na implementação do sistema e integra o serviço Apache para execução como sistema web. Os demais aspectos do arquivo Dockerfile para o contêiner da aplicação contemplam a instalação e configuração das dependências necessárias para a interação com o serviço de banco de dados, além da utilização adequada do arquivo de configuração PHP para o ambiente de produção. Somado a isso, as atividades de análise de serviços, análise de volumes e definição de cenário guiaram o planejamento do arquivo docker-compose.yml para orquestração do contêiner da aplicação e do contêiner do banco de dados. O cenário adotado foi o Cenário 1. As figuras 14 e 15 demonstram o resultado das atividades de implementação dos arquivos.

Figura 14 – docker-compose.yml para IVET

```yaml
services:
  app:
    container_name: ivet
    build:
      context: .
      dockerfile: Dockerfile
    volumes:
      - ./public_html:/var/www/html
    ports:
      - 8002:80
    depends_on:
      - db
    networks:
      - internal

  db:
    container_name: ivet_db
    image: mysql:8.3
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: '${MYSQL_ROOT_PASSWORD}'
      MYSQL_DATABASE: '${MYSQL_DATABASE}'
      MYSQL_USER: '${MYSQL_USER}'
      MYSQL_PASSWORD: '${MYSQL_PASSWORD}'
    command: [ --default-authentication-plugin=mysql_native_password ]
    ports:
      - 3307:3306
    volumes:
      - ./db:/var/lib/mysql
      - ./ivetune_geral:/docker-entrypoint-initdb.d
    networks:
      - internal

networks:
  internal:
    external: true

volumes:
  db:
```

Fonte: O autor

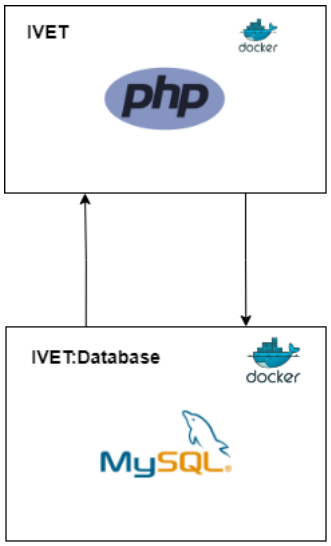
Figura 15 – Dockerfile para IVET

```
FROM php:8.1-apache

COPY public_html/ /var/www/html/

RUN docker-php-ext-configure pdo_mysql --with-pdo-mysql=mysqlnd \
    && docker-php-ext-configure mysqli --with-mysqli=mysqlnd \
    && docker-php-ext-install pdo_mysql \
    && mv "$PHP_INI_DIR/php.ini-production" "$PHP_INI_DIR/php.ini"
```

Fonte: O autor

Nesse contexto, a figura 18 reflete os resultados para o site do G2BC, a tecnologia de implementação não exige instalação de dependências e por isso não se tem um arquivo Dockerfile, apenas um mapeamento de volumes para o conteúdo do site ser refletido no contêiner. As figuras 16 e 17 ilustram as arquiteturas das soluções.

Figura 16 – Arquitetura de contêineres do IVET



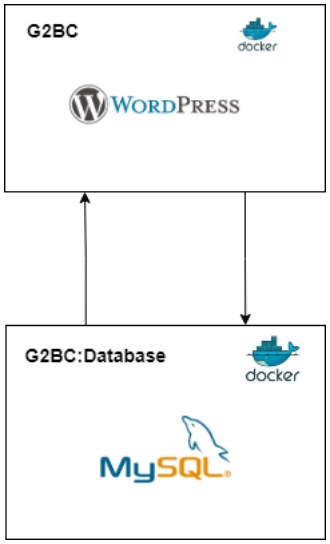
Fonte: O autor

Além disso, foi identificado que a tecnologia de implementação, WordPress, necessitava de instalação adicional no contêiner, mesmo utilizando sua imagem base disponível no Docker Hub [3], o repositório oficial de imagens do Docker. Consequentemente, foi adicionada uma atividade de verificação para esse tipo de tecnologia. Ambas aplicações foram atualizadas em seus repositórios do GitHub com os arquivos de implementação dos contêineres e estão acessíveis através dos endereços https://ivet.uneb.br e https://g2bc.uneb.br.

No contexto do grupo de pesquisa G2BC, é necessário atender ao requisito do servidor Docker na UNEB, onde os contêineres das aplicações, uma vez prontos, rodaram localmente no servidor e, portanto, acessíveis sob um mesmo domínio. Isso levou à configuração de um *proxy*. Dada a natureza do processo, o *proxy* escolhido foi também

[3] https://hub.docker.com

Figura 17 – Arquitetura de contêineres do site G2BC



Fonte: O autor

configurado em um contêiner Docker, o que evidenciou outra necessidade: a criação de uma rede Docker para que os contêineres se reconhecessem pelos seus nomes de *host*, sem a necessidade de investigar os IPs locais gerados por cada contêiner.

O serviço do *proxy* foi implementado em contêiner através do arquivo mostrado na figura 19. Também foram configurados os acessos às aplicações G2BC e IVET no seu arquivo de configuração figura 20. Nas figuras anteriores tambem é possível visualizar a configuração da rede Docker, definida nas seções *networks* nos arquivos docker-compose.yml, comum aos contêineres no ambiente de produção do servidor.

Figura 18 – docker-compose.yml para o site G2BC

```yaml
services:
  app:
    image: wordpress:latest
    container_name: g2bc
    restart: always
    ports:
      - 8001:80
    volumes:
      - ./public_html/wp-content:/var/www/html/wp-content
      - ./wp_data:/var/www/html/
    environment:
      WORDPRESS_DB_HOST: '${WORDPRESS_DB_HOST}:3306'
      WORDPRESS_DB_USER: '${WORDPRESS_DB_USER}'
      WORDPRESS_DB_PASSWORD: '${WORDPRESS_DB_PASSWORD}'
      WORDPRESS_DB_NAME: '${WORDPRESS_DB_NAME}'
    depends_on:
      - db
    networks:
      - internal
  db:
    container_name: '${WORDPRESS_DB_HOST}'
    image: mysql:8.0
    restart: always
    volumes:
      - ./db:/var/lib/mysql
      - ./g2bcun_geral:/docker-entrypoint-initdb.d
    ports:
      - 3306:3306
    expose:
      - 3306
      - 33060
    environment:
      MYSQL_DATABASE: '${MYSQL_DATABASE}'
      MYSQL_USER: '${MYSQL_USER}'
      MYSQL_PASSWORD: '${MYSQL_PASSWORD}'
      MYSQL_ROOT_PASSWORD: '${MYSQL_ROOT_PASSWORD}'
    command: '--default-authentication-plugin=mysql_native_password'
    networks:
      - internal
  phpmyadmin:
    image: phpmyadmin/phpmyadmin
    restart: always
    ports:
      - '3000:80'
    environment:
      PMA_HOST: '${WORDPRESS_DB_HOST}'
      MYSQL_ROOT_PASSWORD: '${MYSQL_ROOT_PASSWORD}'
    depends_on:
      - db
    networks:
      - internal

networks:
  internal:
    external: true

volumes:
  wp_data:
  db:
```

Fonte: O autor

Figura 19 – docker-compose.yml do serviço de *proxy*

```yaml
services:

  haproxy:
    image: haproxy:latest
    container_name: haproxy
    restart: always
    volumes:
      - ./haproxy.cfg:/usr/local/etc/haproxy/haproxy.cfg:ro
      - ./g2bc.uneb.br.pem:/opt/haproxy_install/g2bc.uneb.br.pem
    ports:
      - "80:80"
      - "443:443"
```

Fonte: O autor

Figura 20 – Arquivo de configuração do *proxy*

```
global
    log stdout  format raw  local0  info
    daemon

defaults
    log     global
    mode    http
    option  httplog
    option  dontlognull
    option  http-server-close
    timeout connect 5000
    timeout client  50000
    timeout server  50000

frontend http_front
    bind *:80
    redirect scheme https if { hdr(host) -i g2bc.uneb.br www.g2bc.uneb.br } !{ ssl_fc }
    acl url_g2bc hdr(host) -i www.g2bc.uneb.br g2bc.uneb.br
    acl url_ivet hdr(host) -i www.ivet.uneb.br ivet.uneb.br
    bind :443 ssl crt /opt/haproxy_install/g2bc.uneb.br.pem
    http-request set-header X-Forwarded-Proto https

    use_backend g2bc_backend if url_g2bc
    use_backend ivet_backend if url_ivet

    default_backend g2bc_backend

backend g2bc_backend
    #http-request set-path /
    server g2bc tororao.uneb.br:8001

backend ivet_backend
    #http-request set-path /
    server ivet tororao.uneb.br:8002
```

Fonte: O autor

5.1.2 Ciclo 2

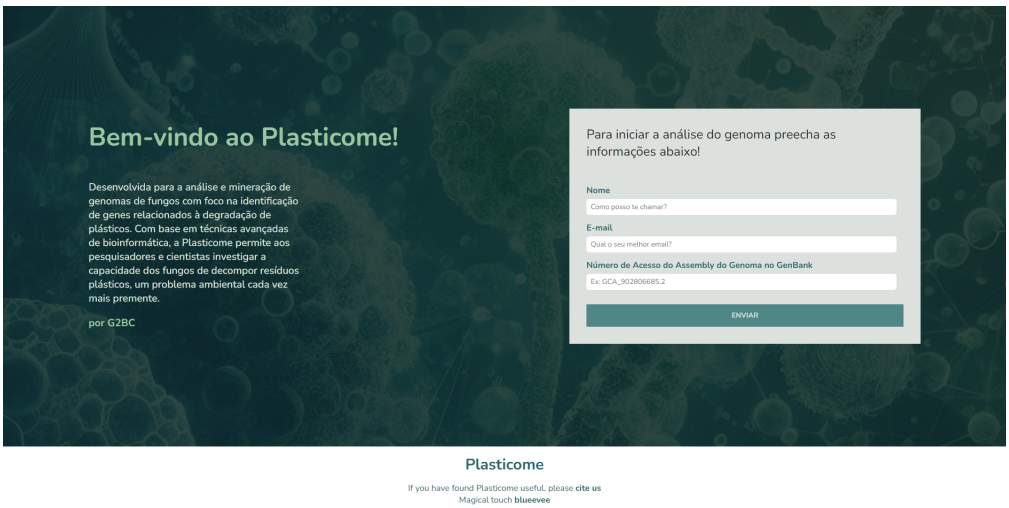
A implementação do ciclo 2 ocorreu sobre a aplicação PLASTICOME, pautada na segunda versão do processo com auxílio da ferramenta de suporte, o Jira. As QMs, PLAS-1: Manutenção adaptativa Docker para configuração do software em servidor; PLAS-6: Ajustes de sintaxe e instruções de usabilidade no *frontend*; e PLAS-7: Configuração para envio de e-mail, foram registrada na ferramenta e o fluxo de trabalho seguido conforme detalhamento presente no anexo B.

- **PLASTICOME (*Fungal Genome Plastic Degradation Explorer*):** A aplicação PLASTICOME foi projetada para a mineração em genomas de fungos, visando identificar enzimas capazes de degradar plásticos. Utilizando algoritmos de bioinformática avançados, essa ferramenta permite a análise sistemática de genes relacionados à degradação de polímeros, contribuindo para a busca de soluções sustentáveis para a poluição por plásticos. É uma aplicação com *Frontend*[4] básico em HTML,CSS,JS + *Backend*[5] *Python* com implementação de pipeline e dependências de serviços e um serviço de metadados[6] também em *Python*. Necessita manutenção adaptativa com

[4] https://github.com/G2BC/plasticome-frontend
[5] https://github.com/G2BC/plasticome-backend
[6] https://github.com/G2BC/plasticome-metadata

foco em conteinerização e configuração em servidor. Possui alto grau de dependências, com agravante do ecossistema *Python* não possuir gerenciador para as mesmas. Também possui interação de serviços, além de banco de dados, como gerenciadores de filas e mensagens e etapas do pipeline que realizam criação de contêineres, logo, alta complexidade. A tela inicial do sistema pode ser vista na figura 21.

Figura 21 – Tela inicial do PLASTICOME



Fonte: O autor

Após o segundo ciclo de implementação, o BioDockFlow se mostrou promissor em lidar com aplicações com um maior grau de complexidade e interação de serviços. A aplicação PLASTICOME necessitou da orquestração de seis contêineres de forma a isolar seus principais serviços, assegurando a interoperabilidade e facilidade na identificação de erros. As atividades e o fluxo de trabalho definidos foram adequados para realização das necessidades de manutenção.

Nesta QM de conteinerização (PLAS-1), a análise do gerenciamento de dependências na tecnologia de implementação evidenciou a necessidade de adotar uma imagem base que permita o uso do Docker internamente no contêiner do *backend*. Esse recurso possibilita a execução do serviço Docker para a criação de contêineres dentro de um contêiner, uma vez que o pipeline desta aplicação é composto por três etapas principais: (1) identificação de Enzimas Ativas em Carboidratos (CAZy) utilizando o software dbCan (v4.0.0); (2) submissão dos resultados filtrados da etapa anterior ao ECPred (v1.1); e (3) alinhamento das sequências de proteínas utilizando o BLAST. Tanto o dbCan quanto o ECPred são executados mediante a criação de seus respectivos contêineres. Adicionalmente, as tarefas registradas no servidor operam de forma assíncrona através do Celery, um gerenciador de filas. Deste modo, o comando de inicialização ao final do Dockerfile foi:

```
celery -A plasticome.config.celery worker -l info --pool=processes
↪ --autoscale=10,3 --concurrency=2 &
```

```
dockerd --tls=false --host=tcp://0.0.0.0:2375
↪ --host=unix:///var/run/docker.sock &
cd /app && flask run -h backend
```

contemplando a ativação do serviço Docker dentro do contêiner, o registro das tarefas no Celery com as respectivas configurações dos *workers* e, por fim, a inicialização do servidor. O Dockerfile resultante está na figura 22.

Figura 22 – Dockerfile do PLASTICOME backend

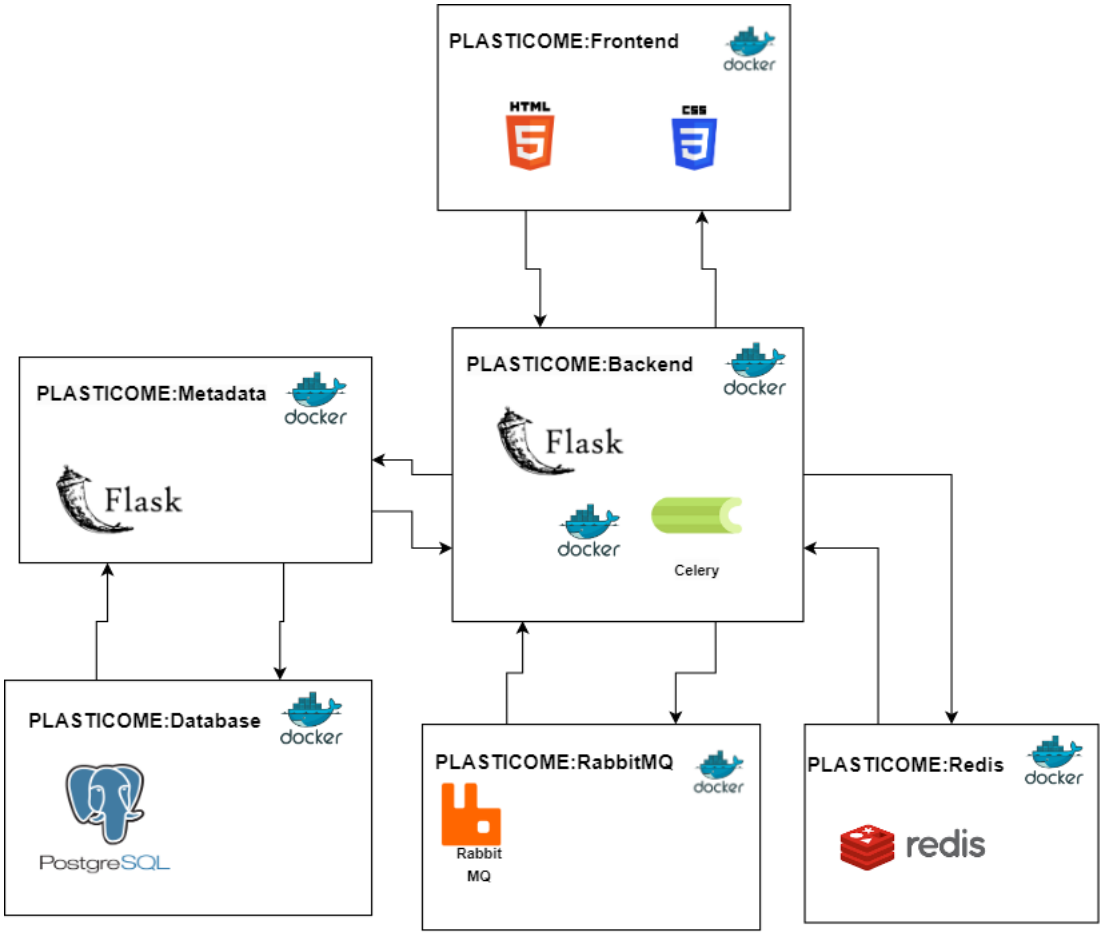
```
FROM docker:24.0.2-dind as builder
USER root
COPY . /app
WORKDIR /app

RUN apk update && apk add --no-cache \
  ca-certificates \
  curl \
  bash \
  tzdata \
  build-base \
  python3 \
  python3-dev \
  libffi-dev \
  openssl-dev \
  musl-dev \
  gcc \
  make \
  docker-cli

RUN curl -o ncbi-blast.tar.gz https://ftp.ncbi.nlm.nih.gov/blast/executables/blast+/LATEST/ncbi-blast-2.16.0+-x64-linux.tar.gz && \
  tar -xzvf ncbi-blast.tar.gz && \
  rm ncbi-blast.tar.gz
ENV BLAST_PATH="/app/ncbi-blast-2.16.0+/bin:${BLAST_PATH}"

RUN curl -o fam-substrate-mapping-08012023.tsv https://bcb.unl.edu/dbCAN2/download/Databases/fam-substrate-mapping-08012023.tsv && \
  mv fam-substrate-mapping-08012023.tsv fam-substrate-mapping.tsv

ENV POETRY_NO_INTERACTION=1 \
  POETRY_VIRTUALENVS_CREATE=false \
  POETRY_CACHE_DIR='/var/cache/pypoetry' \
  POETRY_HOME='/usr/local' \
  POETRY_VERSION=1.5.1

RUN curl -sSL https://install.python-poetry.org | python3 -
RUN poetry install --no-root
RUN chmod +x /app/start.sh
RUN dos2unix /app/start.sh
RUN chmod +x /app/run_flask.sh
RUN dos2unix /app/run_flask.sh
ENV FLASK_APP=plasticome.routes.app.py
FROM builder as runner
CMD [ "/bin/bash", "/app/start.sh" ]
```

Fonte: O autor

No docker-compose.yml, foram configurados contêineres para os serviços do *backend*, *frontend*, serviço de metadados, banco de dados, além de dois serviços para o gerenciamento de mensagens: RabbitMQ e Redis, que atuam em conjunto com o Celery, este por sua vez é executado junto a API no serviço do *backend*. O RabbitMQ é utilizado como *broker* de mensagens, responsável por gerenciar as tarefas assíncronas entre a API e o Celery. Já o Redis é configurado como uma forma de armazenamento dos resultados das tarefas processadas.

Para garantir a comunicação e identificação correta dos nomes definiu-se uma rede Docker específica para o sistema que levou o mesmo nome do software, interligando os serviços. A arquitetura de contêineres que ilustra o docker-compose.yml resultante das atividades pode ser visto na figura 23.

Figura 23 – Arquitetura de contêineres do PLASTICOME



Fonte: O autor

PLASTICOME tem seu código-fonte acessível através dos seus repositórios no GitHub do G2BC, com os arquivos de implementação dos contêineres, e está disponível no endereço https://plasticome.uneb.br.

5.1.2.1 Avaliação quantitativa

Neste ciclo, definiu-se a avaliação quantitativa com o objetivo de avaliar a eficácia do processo por meio da taxa de resolução das QMs com respeito às diferentes rotas de execução do fluxo do processo (1. Docker, 2. Refatoração, 3. Correção de bugs, 4. Solicitação de mudanças, e 5. Documentação). A análise é realizada do ponto de vista dos membros do grupo de pesquisa G2BC, no contexto da manutenção de aplicações web heterogêneas de bioinformática desenvolvidas pelo grupo. A questão a ser respondida é: os fluxos e as atividades do processo promovem a resolução das QMs de acordo com as necessidades do grupo G2BC?

É importante destacar que essa questão de validação se diferencia da questão norteadora do projeto. Enquanto a questão norteadora tem um caráter mais amplo, a

questão de validação é mais restrita, com o propósito de orientar a métrica da avaliação quantitativa. Assim, ela contribui diretamente para a resolução da questão norteadora.

A métrica de avaliação definida foi a Taxa de Resolução de Questões de Manutenção (TR-QM). A TR-QM é calculada pela seguinte fórmula:

$$\text{TR-QM} = \frac{(N^\circ \text{ QM-PA Concluídas} \times 1) + (N^\circ \text{ QM-PM Concluídas} \times 0,5) + (N^\circ \text{ QM-PB Concluídas} \times 0,25)}{(Total \text{ QM-PA} \times 1) + (Total \text{ QM-PM} \times 0,5) + (Total \text{ QM-PB} \times 0,25)} \tag{5.1}$$

As QMs foram classificadas em cinco categorias: 1. Docker, 2. Refatoração, 3. Correção de bugs, 4. Solicitação de mudanças, e 5. Documentação.

Cada QM recebe um peso distinto, atribuído de acordo com sua prioridade e natureza:

- **QM-PA (Prioridade Alta)**: inclui todas as questões da categoria 1 (Docker), com peso atribuído de 1.
- **QM-PM (Prioridade Média)**: abrange questões das categorias 2 (Refatoração) e 3 (Correção de bugs) que impedem o uso do sistema, com peso atribuído de 0,5.
- **QM-PB (Prioridade Baixa)**: inclui questões das categorias 2 (Refatoração) e 3 (Correção de bugs) que não impedem o uso do sistema, bem como das categorias 4 (Solicitação de mudanças) e 5 (Documentação), com peso atribuído de 0,25.

Para o PLASTICOME, além da QM de conteinerização (PLAS-1) houve mais uma do tipo de manutenção perfectiva para ajustes de sintaxe em textos e melhorias de intruções de usabilidade no *frontend* (PLAS-6) e uma do tipo corretiva para adição das configurações de envio de e-mail, função que faz parte do funcionalidade principal da aplicação e compõe a última etapa do pipeline. Nesse contexto, para o cálculo estipulado da métrica de avaliação, temos três QMs: a primeira do tipo de manutenção adaptativa sendo categoria 1 (Docker), a segunda uma manutenção perfectiva da categoria 4 (Solicitação de mudanças) e a terceira uma manutenção corretiva da categoria 3 (Correção de bugs). A seguir, foi realizado o cálculo da métrica a partir das quantidades de QMs na fórmula:

$$\text{TR-QM} = \frac{(1 \times 1) + (1 \times 0,5) + (1 \times 0,25)}{(1 \times 1) + (1 \times 0,5) + (1 \times 0,25)} \tag{5.2}$$

$$\text{TR-QM} = \frac{(1) + (0,5) + (0,25)}{(1) + (0,5) + (0,25)} \tag{5.3}$$

$$\text{TR-QM} = 1 \tag{5.4}$$

A TR-QM obtida foi de valor 1 (um), indicando a capacidade de resolução de QMs de prioridade alta pelo fluxo de trabalho proposto no BioDockFlow, auxiliado pela ferramenta Jira.

5.1.3 Ciclo 3

O ciclo 3 ocorreu com a implementação do processo por parte da equipe de manutenção do projeto BEM (*Brazilian Edible Mushrooms*). A equipe utilizou a 2ª versão do BioDockFlow[7], bem como o Jira para gerenciamento das QMs. Foram cadastradas um total de 49 QMs, a tabela 1 apresenta um resumo do número de QMs cadastradas distribuidas entre as categorias. As necessidades da equipe levaram a adaptabilidade do processo durante o seu uso resultando na 3ª versão do BioDockFlow, presente nos anexos B e C e no respositório do GitHub[8].

Tabela 1 – Quantidade de QMs por categoria no projeto BEM

Categoria	Quantidade de QMs
CORREÇÃO DE BUGS	12
DOCKER -> CENÁRIO 1	1
MUDANÇA DE REQUISITOS	12
NOVOS REQUISITOS	20
REFATORAÇÃO	4
Total	49

Fonte: O autor

- **Projeto BEM (*Brazilian Edible Mushrooms*):** O projeto BEM se trata de um sistema voltado para visualização e gerenciamento de registros dos cogumelos identificados como comestíveis no brasil. É uma aplicação com *frontend*[9] ReactJS + *backend*[10] Laravel; essa aplicação necessita de todos os tipos de manutenção, apresenta necessidade de implementação de requisitos funcionais, adaptabilidade ao servidor, correção de erros e melhorias na usabilidade; possui uma arquitetura um pouco mais robusta por ser construído em *frameworks* tanto no *frontend* quanto *backend*, mas não apresenta implementação de pipeline e conta apenas com um serviço adicional de banco de dados, portanto, média complexidade. A página inicial da aplicação pode ser vista na figura 24.

[7] https://github.com/G2BC/BioDockFlow/tree/main/v2
[8] https://github.com/G2BC/BioDockFlow
[9] https://github.com/G2BC/BEM-web
[10] https://github.com/G2BC/BEM-server

Figura 24 – Tela inicial do BEM

Fonte: O autor

Para a resolução da QM de conteinerização (BEM-40) foram implementados dois Dockerfiles, devido a robustez do *frontend* e *backend*, ambos implementados em *frameworks* e, portanto, contendo dependências que precisam ser provisionadas através desses arquivos. As figuras 25 e 26 demonstram os Dockerfiles implementados.

Figura 25 – Dockerfile do *backend* BEM

```
FROM php:8.1-fpm

RUN apt-get update && apt-get install -y \
    build-essential \
    libpng-dev \
    libjpeg-dev \
    libfreetype6-dev \
    locales \
    zip \
    jpegoptim optipng pngquant gifsicle \
    vim \
    unzip \
    git \
    curl \
    libpq-dev \
    netcat-traditional

RUN apt-get clean && rm -rf /var/lib/apt/lists/*

ADD --chmod=0755 https://github.com/mlocati/docker-php-extension-installer/releases/latest/download/install-php-extensions /usr/local/bin/
RUN install-php-extensions bz2 gd gettext exif pdo_pgsql grpc pgsql zip
RUN docker-php-ext-enable pdo_pgsql pgsql grpc exif gettext gd bz2 zip
RUN curl -sS https://getcomposer.org/installer | php -- --install-dir=/usr/local/bin --filename=composer

COPY . /var/www
WORKDIR /var/www

RUN chmod -R 777 storage/
RUN composer i
RUN php artisan jwt:secret

EXPOSE 9000
```

Fonte: O autor

Durante a análise dos serviços foi identificado que o Laravel é, em síntese, uma aplicação PHP e não inclui um serviço web para lidar com as requisições de rede, nesse âmbito, para o provisionamento do *backend* foi necessário aliar o contêiner construído através da Dockerfile da figura 25, baseado na imagem Docker php:8.1-fpm que é mais indicada para gerenciar o processamento de requisições e a execução de processos PHP em segundo plano, a um contêiner contendo um serviço web Nginx. Nesse contexto, o Nginx

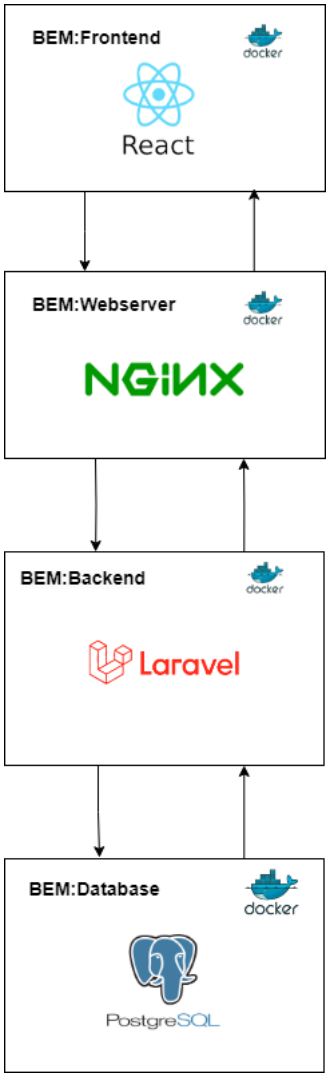
Figura 26 – Dockerfile do *frontend* BEM

```
1   FROM node:21.7.2
2
3   WORKDIR /root/bem-web
4
5   COPY . .
6   COPY package.json ./
7   COPY package-lock.json ./
8
9   RUN npm install
10  RUN npm run build
11
12  EXPOSE 3000
13
14  CMD ["npm", "start"]
```

Fonte: O autor

atua como um *proxy* reverso, direcionando as requisições para o contêiner PHP-FPM, que processa o código PHP. A figura 27 ilustra a arquitetura de contêineres da solução.

Figura 27 – Arquitetura de contêineres do BEM



Fonte: O autor

O código-fonte do sistema BEM é acessível através dos seus repositórios no GitHub do G2BC e está disponível no endereço https://bem.uneb.br.

5.1.3.1 Avaliação quantitativa

Nesse contexto, das 49 QMs cadastradas 21 foram concluídas, seguindo a distribuição entre as categorias apresentadas na tabela 2.

Tabela 2 – Quantidade de QMs concluídas por categoria no projeto BEM

Categoria	Quantidade de QMs
CORREÇÃO DE BUGS	7
DOCKER -> CENÁRIO 1	1
MUDANÇA DE REQUISITOS	10
NOVOS REQUISITOS	3
REFATORAÇÃO	0
Total	21

Fonte: O autor

Logo, de acordo com a tabela 1 apresentada anteriormente, obteve-se para o cálculo da TR-QM do projeto BEM:

$$\text{TR-QM} = \frac{(1 \times 1) + (7 \times 0,5) + (13 \times 0,25)}{(1 \times 1) + (16 \times 0,5) + (32 \times 0,25)} \tag{5.5}$$

$$\text{TR-QM} = \frac{(1) + (3{,}5) + (3{,}25)}{(1) + (8) + (8)} \tag{5.6}$$

$$\text{TR-QM} = \frac{(7{,}75)}{(17)} \tag{5.7}$$

$$\text{TR-QM} \approx 0{,}45 \tag{5.8}$$

A TR-QM obtida foi de valor aproximado de 0,45, ou seja, 45% de eficácia do processo na resolução das QMs do projeto BEM. Se faz importante salientar que o projeto está em andamento e algumas QMs estão aguardando a disponibilidade da equipe de desenvolvimento para serem solucionadas. O valor da TR-QM para foi cálculado apenas para feitos demonstrativos, uma vez que o processo de manutenção ainda está em curso e não há impedimentos para execução das QMs. A TR-QM é obtida, de forma definitiva, ao final do cronograma de manutenção realizado pela equipe, quando a aplicação será de fato disponibilizada aos usuários.

5.1.3.2 Avaliação qualitativa

A avaliação qualitativa do BioDockFlow teve como objetivo analisar a experiência de seus usuários no contexto da manutenção de software, através de consulta à equipe

responsável pelo projeto BEM. Para isso, foi aplicado um formulário contendo uma questão aberta com o seguinte enunciado: "Descreva detalhadamente a sua experiência de uso do BioDockFlow para manutenção da aplicação BEM."

As respostas coletadas forneceram *insights* relevantes, evidenciando tanto aspectos positivos quanto oportunidades de melhoria. Entre os principais pontos destacados, o primeiro refere-se à aplicabilidade e usabilidade do BioDockFlow para desenvolvedores. Uma das respostas salientou que o documento serve como um guia eficaz para profissionais experientes que atuam de forma individual, mas identificou uma lacuna no suporte às dinâmicas de equipe, especialmente no que tange à criação, revisão e integração de tarefas no repositório do projeto. Essa observação sugere a necessidade de aprimorar a metodologia para atender contextos colaborativos, indicando que, embora robusta para cenários individuais, a abordagem atual pode ser expandida para contextos mais complexos de trabalho em grupo.

O segundo aspecto analisado diz respeito à organização e estruturação de projetos. Uma das respostas descreveu o processo de manutenção do sistema BEM como desafiador, devido à curva de aprendizado associada à introdução de novas tecnologias em sua formação técnica. Contudo, o BioDockFlow destacou-se como um elemento central na organização, oferecendo uma estrutura clara que facilita o alinhamento das ações realizadas. A disponibilização de um fluxo de trabalho bem definido e do documento com as atividades detalhadas foi apontada como um fator essencial para aumentar a eficácia do processo. Adicionalmente, a capacidade do BioDockFlow de incorporar melhorias práticas foi ressaltada, exemplificada pela inclusão das etapas diretamente na ferramenta Jira e pela introdução de uma nova fase para explicitar o momento adequado de realizar o *pull request*, aprimorando o controle e a coordenação entre os integrantes.

O terceiro ponto abordado refere-se aos benefícios estratégicos e ao diagnóstico de melhorias proporcionados pelo BioDockFlow. Uma das respostas enfatizou sua contribuição para a definição clara do "como fazer", destacando a padronização de ferramentas e critérios de qualidade. Essa abordagem promoveu uma experiência de desenvolvimento mais fluida e produtiva, além de possibilitar, em momentos de menor produtividade, a identificação de áreas críticas de aprimoramento, como testes, documentação e comunicação. Esse aspecto reforça o potencial do BioDockFlow como processo de manutenção, refletindo também no diagnóstico e solução de gargalos durante o processo.

De forma geral, as respostas indicam que o BioDockFlow é eficaz em oferecer estrutura e organização, proporcionando benefícios como clareza na definição de etapas, adaptabilidade às necessidades práticas por meio de melhorias iterativas e impacto positivo na produtividade. Entretanto, foram apontadas oportunidades de aprimoramento, como a ampliação do suporte para dinâmicas de trabalho em equipe, particularmente no contexto de revisão e integração de tarefas, demonstrando a necessidade de expandir o processo

para abranger fluxos colaborativos mais complexos.

Portanto, a avaliação qualitativa sugere que o BioDockFlow atende de maneira eficaz a cenários individuais e estruturados, ao mesmo tempo em que apresenta potencial para evolução e aplicação em contextos colaborativos. Assim, consolida-se como uma ferramenta versátil e robusta para a manutenção de software, com possibilidades de adaptação para atender a demandas mais amplas no desenvolvimento e gestão de projetos.

5.2 Avaliação 2

A avaliação 2 teve como objetivo principal analisar a eficácia do processo na garantia da interoperabilidade entre aplicações implantadas em um mesmo servidor, verificando se as aplicações estão disponíveis no ambiente de produção e funcionais. Durante a etapa de implantação, foram enfrentadas diversas dificuldades, desde o manejo adequado de resíduos de testes com contêineres e a adoção de boas práticas para limpeza do ambiente, até questões relacionadas à configuração da rede, com vistas a assegurar a comunicação eficiente entre os contêineres das aplicações e sua acessibilidade externa à rede interna da UNEB.

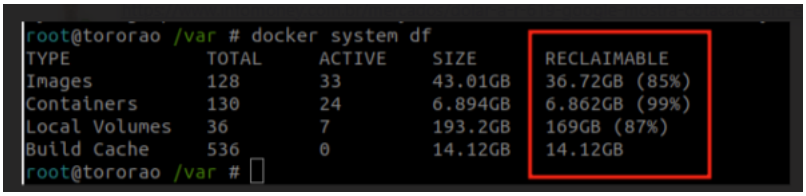
A principal dificuldade identificada foi a configuração do serviço de *proxy*. Ao disponibilizar várias aplicações sob um mesmo domínio, tornou-se essencial garantir que o serviço fosse configurado adequadamente para redirecionar as requisições aos contêineres corretos, utilizando os endereços URL como referência. Inicialmente, as aplicações foram disponibilizadas por meio do protocolo HTTP. Posteriormente, a adição de certificados digitais possibilitou a migração para o protocolo HTTPS, que oferece maior autenticidade e segurança às conexões. No entanto, essa mudança inviabilizou a comunicação entre o *frontend* e *backend* das aplicações PLASTICOME e BEM por meio do protocolo HTTP, tornando necessário configurá-las para operar integralmente sob HTTPS, bem como atribuir URLs específicas a cada *backend* no serviço de *proxy*.

Ademais, os testes realizados no ambiente de contêineres consumiram recursos significativos do servidor. O espaço ocupado por imagens, contêineres, volumes não utilizados e pela memória cache dos *builds* alcançou níveis consideráveis, como pode ser visto na figura 28. Foi criado um *script* com os comandos a seguir para uma rotina semanal de limpeza como uma ação preventiva para liberar recursos:

```
docker container prune
docker system prune -a --volumes
```

Contudo, apesar das dificuldades enfrentadas, não foram identificadas limitações que comprometem a interoperabilidade das aplicações. Todas estão plenamente acessíveis por meio dos endereços https://g2bc.uneb.br, https://ivet.uneb.br, https://plasticome.uneb.br

Figura 28 – Espaço de memória disponível para limpeza dos recursos Docker



Fonte: O autor

e https://bem.uneb.br. Essas aplicações foram implantadas em um único servidor e encontram-se em plena operação, sem que o funcionamento de uma interfira nas demais.

6 CONSIDERAÇÕES FINAIS

Os resultados obtidos cumprem com os objetivos geral e específicos deste trabalho. O processo proposto demonstrou capacidade de orientar os usuários na realização das QMs de conteinerização com foco em garantir a interoperabilidade das aplicações desenvolvidas pelo G2BC, atendendo também às demandas específicas de manutenção por meio de fluxos de trabalho alinhados aos diferentes tipos de manutenção descritos na literatura e da utilização do Jira.

Nesse âmbito, alinhado à metodologia, o BioDockFlow validou as conjecturas teóricas propostas: oferece uma abordagem controlada e estruturada para realizar mudanças, sendo capaz de atender às necessidades específicas das aplicações de bioinformática sem ter sua eficácia e adaptabilidade comprometidas pela heterogeneidade desses sistemas.

Com o BioDockFlow, o G2BC foi capaz de compartilhar softwares científicos previamente fora de operação, viabilizando a utilização de produções científicas que, até então, não alcançavam a comunidade acadêmica. Esse resultado permite ao grupo uma participação mais ativa na comunidade científica, contribuindo para os avanços nas pesquisas em bioinformática. Pode-se concluir que o BioDockFlow proporcionou benefícios significativos tanto ao G2BC quanto à comunidade científica, ao promover a reprodutibilidade de experimentos e estender o ciclo de vida dos softwares da área. Validado no contexto do G2BC, o processo demonstra potencial para ser aplicado por qualquer grupo de pesquisa na área de bioinformática. A versão 3 e atual do BioDockFlow, além das demais versões, está disponível no repositório https://github.com/G2BC/BioDockFlow.

No entanto, a versão do processo resultante do ciclo 3 apresenta limitações no que diz respeito à maturidade para atender fluxos de trabalho de equipes de manutenção compostas por três ou mais desenvolvedores. A superação dessa limitação exige a aplicação do processo por grupos com esse perfil, seguida da execução ativa da última fase, visando à melhoria contínua.

Como propostas para trabalhos futuros, sugere-se a aplicação do processo na manutenção de aplicações web de maior complexidade, incluindo outros softwares desenvolvidos pelo G2BC, como FunRegulation[1,2], FunExpression[3], AGSSA e web VAMPIRE[4,5,6]. Além disso, recomenda-se sua utilização por grupos de manutenção com três ou mais integrantes. Essas iniciativas não apenas visam otimizar e amadurecer o BioDockFlow, mas também

[1] https://github.com/G2BC/funregulation-frontend
[2] https://github.com/G2BC/FunRegulationAPI
[3] https://github.com/G2BC/funexpression
[4] https://github.com/RevolverOcelotIII/GenomeInterface
[5] https://github.com/RevolverOcelotIII/Genome-API
[6] https://github.com/RevolverOcelotIII/Phenotypes_Finder

expandir suas capacidades para abranger diversos contextos de manutenção na área da bioinformática.

Outros fatores a serem explorados estão no âmbito de uma análise mais detalhada e quantitativa sobre métricas de qualidade relacionadas ao processo e ao desempenho do BioDockFlow, alinhando-se às características definidas pela (International Organization for Standardization, 2024), como *performance efficiency* e *reliability*. Métricas como tempo de deploy, taxa de falhas, e desempenho sob diferentes cargas de trabalho poderiam ser avaliadas, incorporando também desafios de escalabilidade para demonstrar a capacidade do sistema de suportar múltiplas aplicações concorrentes em alto volume de dados.

Pode ser trabalhado ainda a abordagem do gerenciamento eficiente de recursos computacionais e a implementação de boas práticas de segurança em contêineres. Ferramentas como Docker Bench for Security e Trivy podem ser utilizadas para demonstrar um *hardening* e a análise de vulnerabilidades. Adicionalmente, podem ser abordadas estratégias para deploy em produção, como orquestração e balanceamento de carga, bem como a criação de um sistema de monitoramento e gerenciamento de logs para as aplicações em contêiner. Isso contribuiria para avaliar melhor o impacto do uso de contêineres no desempenho das aplicações, abrangendo tanto as características de *maintainability* quanto *security* previstas na (International Organization for Standardization, 2024).

REFERÊNCIAS

AFGAN, E. *et al.* The galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2016 update. **Nucleic Acids Research**, v. 44, n. W1, p. W3–W10, 2016. Citado na página 17.

BINI, A. M. *et al.* Refatorando um pipeline de bioinformática: Um estudo de caso para análise de amplicons. In: **Anais do XV Brazilian e-Science Workshop**. Porto Alegre, RS, Brasil: SBC, 2021. p. 137–140. ISSN 2763-8774. Disponível em: https://sol.sbc.org.br/index.php/bresci/article/view/15799. Citado na página 16.

BOETTIGER, C. An introduction to docker for reproducible research. **SIGOPS Oper. Syst. Rev.**, Association for Computing Machinery, New York, NY, USA, v. 49, n. 1, p. 71–79, jan 2015. ISSN 0163-5980. Disponível em: https://doi.org/10.1145/2723872.2723882. Citado nas páginas 12 e 18.

BRUNO, A.; AURY, J.; ENGELEN, S. Boardion: real-time monitoring of oxford nanopore sequencing instruments. **BMC bioinformatics**, England, v. 22, n. 1, p. 245, 5 2021. ISSN 1471-2105 (Electronic). Disponível em: https://pubmed.ncbi.nlm.nih.gov/33985424/. Citado na página 63.

BUCHER, E. *et al.* Annot: a django-based sample, reagent, and experiment metadata tracking system. **BMC Bioinformatics**, v. 20, n. 1, p. 542, Nov 2019. ISSN 1471-2105. Citado na página 64.

CAO, Q. *et al.* Gcgx: transcriptome-wide exploration of the response to glucocorticoids. **Journal of molecular endocrinology**, England, v. 68, n. 2, p. B1–B4, 12 2021. ISSN 1479-6813 (Electronic). Disponível em: https://pubmed.ncbi.nlm.nih.gov/34787097/. Citado na página 63.

CARLUCCI, M. *et al.* Discorhythm: an easy-to-use web application and r package for discovering rhythmicity. **Bioinformatics (Oxford, England)**, England, v. 36, n. 6, p. 1952–4, 11 2019. ISSN 1367-4811 (Electronic). Disponível em: https://pubmed.ncbi.nlm.nih.gov/31702788/. Citado nas páginas 22 e 62.

CASTILLO-LARA, S.; ABRIL, J. Ppaxe: easy extraction of protein occurrence and interactions from the scientific literature. **Bioinformatics (Oxford, England)**, England, v. 35, n. 14, p. 2523–2524, 7 2019. ISSN 1367-4811 (Electronic). Disponível em: https://pubmed.ncbi.nlm.nih.gov/30500875/. Citado na página 65.

CITO, J.; GALL, H. C. Using docker containers to improve reproducibility in software engineering research. In: **Proceedings of the 38th International Conference on Software Engineering Companion**. New York, NY, USA: Association for Computing Machinery, 2016. (ICSE '16), p. 906–907. ISBN 9781450342056. Disponível em: https://doi.org/10.1145/2889160.2891057. Citado nas páginas 12 e 13.

CREED, J. H. *et al.* epitad: a web application for visualizing chromosome conformation capture data in the context of genetic epidemiology. **Bioinformatics**, England, v. 35, n. 21, p. 4462–4464, Nov 2019. ISSN 1367-4811. Disponível em: https://pubmed.ncbi.nlm.nih.gov/31099399/. Citado na página 65.

DEGNAN, D. *et al.* Pspecter: A user-friendly and interactive application for visualizing top-down and bottom-up proteomics data in r. **Journal of proteome research**, United States, v. 20, n. 4, p. 2014–2020, 4 2021. ISSN 1535-3907 (Electronic). Disponível em: https://pubmed.ncbi.nlm.nih.gov/33661636/. Citado nas páginas 22 e 62.

DENECKER, T. *et al.* Pixel: a content management platform for quantitative omics data. **PeerJ**, United States, v. 7, p. e6623, 2019. ISSN 2167-8359 (Print). Disponível em: https://pubmed.ncbi.nlm.nih.gov/30944779/. Citado nas páginas 22 e 64.

GOECKS, L. S. *et al.* Design science research in practice: review of applications in industrial engineering. **Gestão Produção**, Universidade Federal de São Carlos, v. 28, n. 4, p. e5811, 2021. ISSN 0104-530X. Disponível em: https://doi.org/10.1590/1806-9649-2021v28e5811. Citado na página 24.

HOLMER, R. *et al.* Genenotebook, a collaborative notebook for comparative genomics. **Bioinformatics (Oxford, England)**, England, v. 35, n. 22, p. 4779–4781, 11 2019. ISSN 1367-4811 (Electronic). Disponível em: https://pubmed.ncbi.nlm.nih.gov/31199463/. Citado na página 63.

HOPKINS, R.; JENKINS, K. **Eating the IT Elephant: Moving from Greenfield Development to Brownfield**. 1. ed. [S.l.]: IBM Press, 2008. ISBN 0137130120. Citado na página 20.

International Organization for Standardization. **ISO/IEC 25002:2024 - Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models**. Geneva, Switzerland, 2024. Disponível em: https://www.iso.org/standard/78175.html. Citado na página 55.

JAHN, C. *et al.* Cellenium-a scalable and interactive visual analytics app for exploring multimodal single-cell data. **Bioinformatics (Oxford, England)**, England, v. 39, n. 6, 6 2023. ISSN 1367-4811 (Electronic). Disponível em: https://pubmed.ncbi.nlm.nih.gov/37261846/. Citado na página 63.

JANSSON, A.-S. **Software Maintenance and Process Improvement by CMMI**. Dissertação (Tese) — Uppsala University, Almedalen, Suécia, 2007. Citado na página 20.

KAGAMI, L. *et al.* The acpype web server for small-molecule md topology generation. **Bioinformatics (Oxford, England)**, England, v. 39, n. 6, 6 2023. ISSN 1367-4811 (Electronic). Disponível em: https://pubmed.ncbi.nlm.nih.gov/37252824/. Citado na página 64.

KARATZA, E. *et al.* Flame (v2.0): advanced integration and interpretation of functional enrichment results from multiple sources. **Bioinformatics (Oxford, England)**, England, v. 39, n. 8, 8 2023. ISSN 1367-4811 (Electronic). Disponível em: https://pubmed.ncbi.nlm.nih.gov/37540207/. Citado na página 64.

KIMBLE, M. *et al.* medna-metadata: an open-source data management system for tracking environmental dna samples and metadata. **Bioinformatics (Oxford, England)**, England, v. 38, n. 19, p. 4589–4597, 9 2022. ISSN 1367-4811 (Electronic). Disponível em: https://pubmed.ncbi.nlm.nih.gov/35960154/. Citado nas páginas 15 e 64.

KöHLER, N. *et al.* Investigating global lipidome alterations with the lipid network explorer. **Metabolites**, Switzerland, v. 11, n. 8, 7 2021. ISSN 2218-1989 (Print). Disponível em: https://pubmed.ncbi.nlm.nih.gov/34436429/. Citado na página 64.

LEPREVOST, F. da V. *et al.* BioContainers: an open-source and community-driven framework for software standardization. **Bioinformatics**, v. 33, n. 16, p. 2580–2582, 03 2017. ISSN 1367-4803. Disponível em: https://doi.org/10.1093/bioinformatics/btx192. Citado nas páginas 16 e 19.

MAHI, N. *et al.* Grein: An interactive web platform for re-analyzing geo rna-seq data. **Scientific reports**, England, v. 9, n. 1, p. 7580, 5 2019. ISSN 2045-2322 (Electronic). Disponível em: https://pubmed.ncbi.nlm.nih.gov/31110304/. Citado nas páginas 15, 22 e 65.

MANCHANDA, N. *et al.* Genomeqc: a quality assessment tool for genome assemblies and gene structure annotations. **BMC genomics**, England, v. 21, n. 1, p. 193, 3 2020. ISSN 1471-2164 (Electronic). Disponível em: https://pubmed.ncbi.nlm.nih.gov/32122303/. Citado na página 65.

MANGUL, S. *et al.* Challenges and recommendations to improve the installability and archival stability of omics computational tools. **PLOS Biology**, Public Library of Science, v. 17, n. 6, p. 1–16, 06 2019. Disponível em: https://doi.org/10.1371/journal.pbio.3000333. Citado nas páginas 15 e 16.

MILLER, M. A.; PFEIFFER, W.; SCHWARTZ, T. The cipres science gateway: a community resource for phylogenetic analyses. In: **Proceedings of the 2011 TeraGrid Conference: Extreme Digital Discovery**. New York, NY, USA: Association for Computing Machinery, 2011. (TG '11). ISBN 9781450308885. Disponível em: https://doi.org/10.1145/2016741.2016785. Citado na página 17.

O'CONNOR, B. D. *et al.* The dockstore: enabling modular, community-focused sharing of docker-based genomics tools and workflows [version 1; peer review: 2 approved]. **F1000Research**, v. 6, p. 52, 2017. Disponível em: https://doi.org/10.12688/f1000research.10137.1. Citado nas páginas 18 e 19.

OLIVEIRA, P. W. de. **Gerenciamento de proveniência de dados de workflows de bioinformática em ambiente de nuvens federadas**. Dissertação (Dissertação (Mestrado em Informática)) — Universidade de Brasília, Brasília, Brasil, 2019. 84 f., il. Citado na página 18.

PIMENTEL, M.; FILIPPO, D.; SANTOS, T. M. dos. Design science research: pesquisa científica atrelada ao design de artefatos. Revista de Educacao a Distancia e Elearning, v. 3, n. 1, 2020. Disponível em: https://revistas.rcaap.pt/lead_read/article/view/21898. Citado na página 24.

PRESSMAN, R. S.; MAXIM, B. R. **Engenharia de Software: Uma Abordagem Profissional**. 9. ed. Porto Alegre, Brazil: AMGH Editora, 2021. Citado nas páginas 21, 22, 28, 29 e 30.

SANDåS, K. *et al.* Nanometa live: a user-friendly application for real-time metagenomic data analysis and pathogen identification. **Bioinformatics (Oxford, England)**, England, v. 40, n. 3, 3 2024. ISSN 1367-4811 (Electronic). Disponível em: https://pubmed.ncbi.nlm.nih.gov/38407280/. Citado na página 65.

SANSONE, S.-A. *et al.* Toward interoperable bioscience data. **Nature Genetics**, v. 44, n. 2, p. 121–126, Feb 2012. ISSN 1546-1718. Disponível em: https://doi.org/10.1038/ng.1054. Citado nas páginas 12 e 16.

SHAO, D. *et al.* Riboa: a web application to identify ribosome a-site locations in ribosome profiling data. **BMC bioinformatics**, England, v. 22, n. 1, p. 156, 3 2021. ISSN 1471-2105 (Electronic). Disponível em: https://pubmed.ncbi.nlm.nih.gov/33765913/. Citado nas páginas 22 e 62.

SILVER, A. Software simplified. **Nature**, v. 546, n. 7656, p. 173–174, 2017. ISSN 1476-4687. Disponível em: https://doi.org/10.1038/546173a. Citado nas páginas 12 e 18.

SOMMERVILLE, I. **Engenharia de Software**. 9. ed. São Paulo, Brazil: Pearson, 2011. Citado nas páginas 19, 20, 22 e 31.

TAM, J. Z. *et al.* A containerization framework for bioinformatics software to advance scalability, portability, and maintainability. In: **Proceedings of the 14th ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics**. New York, NY, USA: Association for Computing Machinery, 2023. (BCB '23). ISBN 9798400701269. Disponível em: https://doi.org/10.1145/3584371.3612948. Citado nas páginas 12, 18 e 62.

URIARTE, R.; HERRERA-NIETO, P. Evam-tools: tools for evolutionary accumulation and cancer progression models. **Bioinformatics (Oxford, England)**, England, v. 38, n. 24, p. 5457–5459, 12 2022. ISSN 1367-4811 (Electronic). Disponível em: https://pubmed.ncbi.nlm.nih.gov/36287062/. Citado nas páginas 23 e 62.

VELDE, K. van der *et al.* Molgenis research: advanced bioinformatics data software for non-bioinformaticians. **Bioinformatics (Oxford, England)**, England, v. 35, n. 6, p. 1076–1078, 3 2019. ISSN 1367-4811 (Electronic). Disponível em: https://pubmed.ncbi.nlm.nih.gov/30165396/. Citado nas páginas 15 e 63.

VOLANT, S. *et al.* Shaman: a user-friendly website for metataxonomic analysis from raw reads to statistical analysis. **BMC bioinformatics**, England, v. 21, n. 1, p. 345, 8 2020. ISSN 1471-2105 (Electronic). Disponível em: https://pubmed.ncbi.nlm.nih.gov/32778056/. Citado na página 65.

WRATTEN, L.; WILM, A.; GöKE, J. Reproducible, scalable, and shareable analysis pipelines with bioinformatics workflow managers. **Nature Methods**, v. 18, n. 10, p. 1161–1168, Oct 2021. Epub 2021 Sep 23. Citado na página 17.

Anexos

ANEXO A – PLANILHA-RESUMO DA REVISÃO DE LITERATURA

Descrição	Avaliação
Apresenta um framework para conteinerização utilizando como exemplo uma aplicação web que produz visualizações 3D dos arquivos de entrada e saída utilizando Frontend NGL viewer + WebServer NodeJS + Software de linha de comando (Tam *et al.*, 2023).	Relevante, apresenta seção descritiva sobre a conteinerização da aplicação.
Apresenta uma aplicação web R e pacote R/Bioconductor para estimar fase, amplitude e significância estatística usando quatro abordagens populares para caracterizar a ritmicidade em dados biológicos temporais(Carlucci *et al.*, 2019).	Baixa relevância, não há comentários sobre a conteinerização.
Apresenta uma aplicação web R Shiny para controle de qualidade e pesquisa de dados proteômicos (Degnan *et al.*, 2021).	Relevante, apresenta seção descritiva sobre a conteinerização da aplicação com interação entre três contêineres compartilhando o mesmo volume.
Apresenta uma aplicação web que identifica a localização mais precisa do sítio A em um fragmento de mRNA protegido por ribossomo e gera os perfis de densidade de leitura do sítio A utilizando Django Python, Nginx, Redis, Celery e PostgreSQL (Shao *et al.*, 2021).	Relevante, apresenta seção descritiva sobre a conteinerização da aplicação com interação entre cinco contêineres, um para cada serviço.
Apresenta uma aplicação web R para modelos de progressão do câncer de ponta e, de forma mais geral, modelos evolutivos de acumulação de eventos (Uriarte; Herrera-Nieto, 2022).	Baixa relevância, não há comentários sobre a conteinerização.

Continua na próxima página

Tabela 3 – continuação da página anterior

Descrição	Avaliação
Apresenta Uma aplicação web R que permite aos pesquisadores visualizar rapidamente as mudanças na abundância de transcritos em resposta aos glicocorticoides em uma variedade de células e espécies (Cao *et al.*, 2021).	Baixa relevância, não há comentários sobre a conteinerização.
Apresenta aplicação web para permitir que pesquisadores experimentais e computacionais consultem, naveguem, visualizem e façam curadoria dos resultados da análise bioinformática para múltiplos genomas utilizando Node.js, MongoDB e NCBI BLAST (Holmer *et al.*, 2019).	Baixa relevância, não há comentários sobre a conteinerização.
Apresenta uma aplicação web R com etapa de preprocessamento de arquivos em C++ para analisar a eficiência das execuções de sequenciamento da ONT (Oxford Nanopore Technologies) (Bruno; Aury; Engelen, 2021).	Baixa relevância, não há comentários sobre a conteinerização.
Apresenta MOLGENIS Research, uma aplicação web para coletar, gerenciar, analisar, visualizar e compartilhar conjuntos de dados biomédicos grandes e complexos. Tecnologias: Java 1.8 com suporte do framework Spring MVC, Apache Maven, Apache Tomcat, PostgreSQL, ElasticSearch, Bootstrap + Vue e Freemarker (Velde *et al.*, 2019).	Baixa relevância, não há comentários sobre a conteinerização.
Apresenta uma aplicação web que permite aos usuários integrar semanticamente e organizar todos os seus estudos de sequenciamento de RNA, ATAC e CITE de células únicas utilizando PostgreSQL, Python 3, GraphQL, ReactJS, TypeScript e Mantine CSS (Jahn *et al.*, 2023).	Baixa relevância, não há comentários sobre a conteinerização.

Continua na próxima página

Tabela 3 – continuação da página anterior

Descrição	Avaliação
Apresenta uma ferramenta baseada em Python Django que utiliza o Antechamber para gerar topologias para compostos químicos e se integra com outras aplicações Python como CCPN e ARIA (Kagami *et al.*, 2023).	Baixa relevância, não há comentários sobre a conteinerização.
Apresenta uma aplicação que oferece aos pesquisadores uma maneira intuitiva de anotar, armazenar, explorar e minerar seus resultados de multiômica utilizando Python Django, PostgreSQL e Nginx (Denecker *et al.*, 2019).	Relevante, apresenta seção descritiva sobre a conteinerização da aplicação com interação entre três contêineres, um para cada serviço.
Apresenta um sistema de gerenciamento de dados para rastrear amostras de DNA ambiental desde a coleta de dados de campo até as análises bioinformáticas utilizando Python, PostgreSQL and PostGIS, RabbitMQ, Celery, Gunicorn, NGINX (Kimble *et al.*, 2022).	Baixa relevância, não há comentários sobre a conteinerização porém a Dockerfile indica interação entre seis contêineres, um para cada serviço.
Apresenta uma aplicação web R para visualização e interpretação de resultados de análise de enriquecimento funcional e literário de múltiplos conjuntos (Karatza *et al.*, 2023).	Baixa relevância, não há comentários sobre a conteinerização.
Apresenta uma aplicação web para calcular e visualizar redes metabólicas lipídicas com propriedades estatísticas utilizando Python Django, PostgreSQL e Nginx (Köhler *et al.*, 2021).	Baixa relevância, não há comentários sobre a conteinerização porém a Dockerfile indica interação entre três contêineres, um para cada serviço.
Oferece uma solução robusta para anotar amostras, reagentes e protocolos experimentais para ensaios estabelecidos nos quais múltiplos cientistas de laboratório estão envolvidos utilizando Python Django, PostgreSQL, Nginx e Debian (Bucher *et al.*, 2019).	Relevante, apresenta seção descritiva sobre a conteinerização da aplicação com interação entre três contêineres, um para cada serviço.

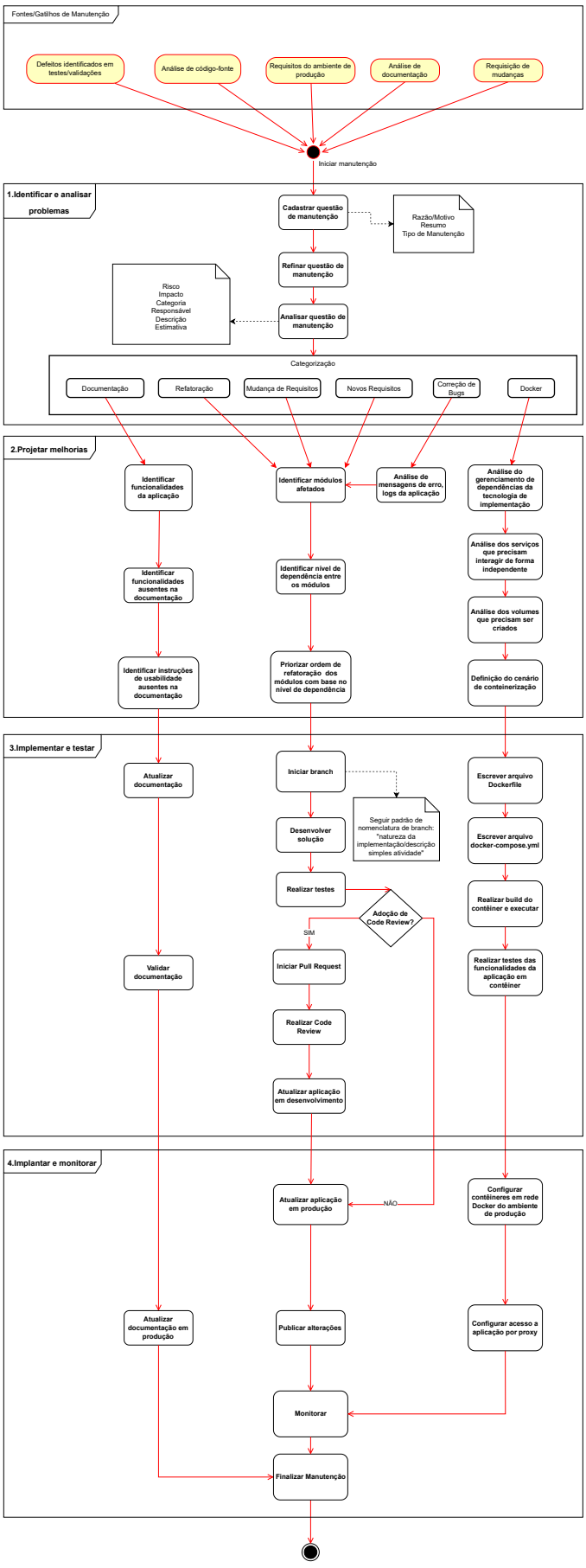
Continua na próxima página

Tabela 3 – continuação da página anterior

Descrição	Avaliação
Apresenta aplicação web R desenvolvida para facilitar o acesso e a análise de dados de RNA-seq depositados no Gene Expression Omnibus (GEO) e Sequence Read Archive (SRA) (Mahi *et al.*, 2019).	Relevante, apresenta seção descritiva sobre a conteinerização da aplicação com interação entre dois contêineres, um para aplicação principal outro para um pacote R com pipeline.
Apresenta um framework web para calcular métricas de contiguidade e completude para montagens genômicas e anotações utilizando R, R Shiny, Python e Shell Script (Manchanda *et al.*, 2020).	Relevante, apresenta seção descritiva sobre a conteinerização.
Apresenta uma aplicação web para análise em tempo real de sequenciamento metagenômico utilizando Python Dash (Sandås *et al.*, 2024).	Baixa relevância, não há comentários sobre a conteinerização.
Apresenta aplicação web R para facilitar o uso de um fluxo de trabalho bioinformático para análise metataxonômica, modelagem estatística confiável e fornecer o maior painel de visualizações interativas entre as aplicações atualmente disponíveis (Volant *et al.*, 2020).	Baixa relevância, não há comentários sobre a conteinerização.
Apresenta uma aplicação web baseada em Python com versão linha de comando que permite aos usuários extrair interações proteína-proteína (PPis) e ocorrência de proteínas a partir de um conjunto dado de artigos do PubMed e PubMedCentral (Castillo-Lara; Abril, 2019).	Baixa relevância, não há comentários sobre a conteinerização.
Apresenta uma aplicação web R que permite aos pesquisadores comparar a organização genômica 3D e anotações em múltiplos bancos de dados de forma interativa para facilitar a descoberta in silico (Creed *et al.*, 2019).	Baixa relevância, não há comentários sobre a conteinerização.

Fonte: O autor

ANEXO B – FLUXO DE TRABALHO BIODOCKFLOW 3ª VERSÃO


Fontes/Gatilhos de Manutenção
Defeitos identificados em testes/validações
Análise de código-fonte
Requisitos do ambiente de produção
Análise de documentação
Requisição de mudanças
Iniciar manutenção
1.Identificar e analisar problemas
Cadastrar questão de manutenção
Razão/Motivo
Resumo
Tipo de Manutenção
Refinar questão de manutenção
Risco
Impacto
Categoria
Responsável
Descrição
Estimativa
Analisar questão de manutenção
Categorização
Documentação
Refatoração
Mudança de Requisitos
Novos Requisitos
Correção de Bugs
Docker
2.Projetar melhorias
Identificar funcionalidades da aplicação
Identificar módulos afetados
Análise de mensagens de erro, logs da aplicação
Análise do gerenciamento de dependências da tecnologia de implementação
Identificar funcionalidades ausentes na documentação
Identificar nível de dependência entre os módulos
Análise dos serviços que precisam interagir de forma independente
Análise dos volumes que precisam ser criados
Identificar instruções de usabilidade ausentes na documentação
Priorizar ordem de refatoração dos módulos com base no nível de dependência
Definição do cenário de conteinerização
3.Implementar e testar
Atualizar documentação
Iniciar branch
Seguir padrão de nomenclatura de branch: "natureza da implementação/descrição simples atividade"
Escrever arquivo Dockerfile
Desenvolver solução
Escrever arquivo docker-compose.yml
Realizar testes
Adoção de Code Review?
SIM
Realizar build do contêiner e executar
Validar documentação
Iniciar Pull Request
Realizar testes das funcionalidades da aplicação em contêiner
Realizar Code Review
Atualizar aplicação em desenvolvimento
4.Implantar e monitorar
Atualizar aplicação em produção
NÃO
Configurar contêineres em rede Docker do ambiente de produção
Atualizar documentação em produção
Publicar alterações
Configurar acesso a aplicação por proxy
Monitorar
Finalizar Manutenção

ANEXO C – DESCRIÇÃO DE ATIVIDADES BIODOCKFLOW 3ª VERSÃO

Autor: Pedro Victor Santana Benevides

BioDockFlow

Índice

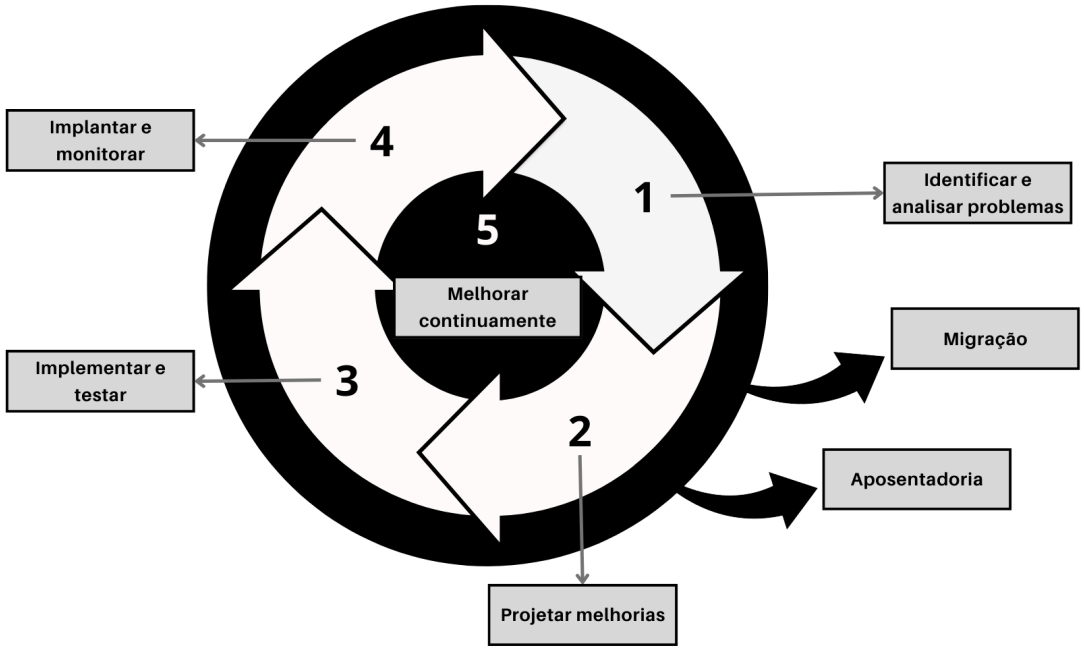
Objetivo

Este processo tem por objetivo gerenciar todos os eventos relacionados aos requisitos do produto a ser desenvolvido pelo projeto. Dentro do escopo de gerenciar destacam-se a definição das fases, especificação das atividades que compõem essas fases e definição da ferramenta de apoio.

Monitoramento do processo

É fundamental o uso de uma ferramenta de apoio para monitoramento das questões de manutenção (QM), validação dos atributos das QMs desde a primeira fase e acompanhamento da sua passagem correta até a conclusão. Para esse processo será utilizada a ferramenta Jira. Deve ser configurado o fluxo de trabalho de acordo com as fases do processo.

Fases do processo



A fase 1, identificar e analisar problemas, é acionada por diversas fontes, como a análise do código-fonte, a revisão da documentação, a identificação de defeitos provenientes de testes ou validações em homologação, solicitações de mudanças pelos usuários e mudanças emergentes devido a alterações no ambiente de produção. As questões levantadas serão categorizadas: problemas identificados em testes e validações podem ser classificados como correção de bugs, novas solicitações de usuários como novos requisitos ou mudança de requisitos, a análise do código-fonte pode indicar a necessidade de refatoração, a revisão da documentação pode apontar a necessidade de atualização, e questões relacionadas ao ambiente de produção podem sugerir a necessidade de conteinerização com Docker para tornar o software mais independente. No entanto, a definição da categoria só poderá ser feita com precisão após uma análise detalhada, que identificará o risco e o impacto em outros módulos do software.

A segunda fase, projetar melhorias, envolve o planejamento da solução com base na categoria definida na fase anterior. Cada categoria possui atividades de planejamento e elaboração específicas, porém, o foco

principal aqui é identificar o que será feito e como será realizado. Nessa fase, são realizadas análises das condições em que a manutenção ocorrerá, bem como a avaliação de logs e mensagens que possam auxiliar na solução dos problemas. Dependendo das necessidades, a equipe de manutenção pode optar por trabalhar em questões de manutenção de forma isolada ou abordar um conjunto de questões simultaneamente, projetando-as juntas e desenvolvendo-as em paralelo, dentro de um mesmo período, para que esse conjunto forme uma entrega única.

Após a fase de projetar melhorias, a terceira fase envolve a implementação da solução, seguindo o escopo definido nas fases anteriores. Neste estágio, a solução é desenvolvida, testada e integrada ao software conforme o planejamento estabelecido. É fundamental que cada responsável por uma questão de manutenção realize testes de funcionalidade ao final de sua atividade, a fim de garantir que a implementação foi correta, ou seja, que não gerou novos pontos de manutenção em potencial e não impactou negativamente outros módulos do sistema.

A quarta fase engloba as atividades de deploy, como a atualização da versão do software em seu repositório, a preparação do ambiente de produção e a disponibilização do sistema para os usuários. No contexto do G2BC, devido à infraestrutura oferecida pelo servidor da UNEB, essa fase envolve uma colaboração entre a equipe de manutenção e uma equipe especializada em infraestrutura e redes. Juntas, elas realizam a solicitação de URLs, a configuração de certificados digitais e a configuração do proxy no servidor para tornar as aplicações acessíveis através das URLs fornecidas. Além disso, o software deve ser monitorado regularmente para garantir a otimização de desempenho e seu pleno funcionamento no ambiente de produção.

A melhoria contínua representa a última fase do processo e envolve a revisão regular e a atualização do sistema, com o objetivo de manter a otimização de desempenho. Essa fase é aplicada tanto ao software quanto ao processo em si, considerando que estratégias específicas de manutenção podem ser desenvolvidas, revisadas e ajustadas conforme as necessidades ao final de cada ciclo de manutenção. Caso um conjunto de questões de manutenção seja abordado, ao término do período estipulado para a implementação de todas as atividades, será realizada uma reunião com a equipe de manutenção para identificar os pontos de melhoria e adaptar o processo ao contexto da equipe.

Fluxo de Trabalho

Atividades

Fase: • **1-Identificar e analisar problemas**
Atividade: **Cadastrar questão de manutenção**
Descrição: Registro da questão de manutenção engatilhada por alguma das fontes.
Pré-condições: • Necessidade de manutenção evidenciada por alguma fonte/motivo
Entradas: • N/A
Tarefas: • Cadastrar questão de manutenção na ferramenta de apoio do processo informando uma descrição da razão/motivo, breve resumo e o tipo de manutenção a ser realizado.
Ferramenta: • Criar item no quadro do projeto na guia da primeira fase
Saídas: • QM - Questão de Manutenção registrada
Observações: • N/A

Fase: • **1-Identificar e analisar problemas**
Atividade: **Refinar questão de manutenção**
Descrição: Registro da questão de manutenção engatilhada por alguma das fontes.
Pré-condições: • N/A
Entradas: • QM registrada no Jira

Tarefas: • Complementar a descrição da QM para incluir exatamente o que precisa ser modificado e quais módulos ou arquivos são afetados.
Ferramenta: • Completar registro da QM no Jira com detalhamento da descrição
Saídas: • Questão de Manutenção com descrição detalhada
Observações: • N/A

Fase: • **1-Identificar e analisar problemas**
Atividade: **Analisar questão de manutenção**
Descrição: Analisar o impacto da QM, identificar as possíveis ações e requisitos necessários para completude na realização da manutenção.
Pré-condições: • N/A
Entradas: • QM registrada no Jira
Tarefas: • Avaliar impacto da mudança solicitada, analisando impacto nos requisitos, artefatos técnicos e no projeto (esforço, prazo, escopo, equipe, riscos). • Avaliar se solicitação de mudança interna impacta os requisitos do Cliente visando envolvê-lo no desenvolvimento da manutenção. • Complementar o registro da QM antes do desenvolvimento com as informações de risco, impacto, categoria (documentação, correção de bugs, dockerização, refatoração ou change request) e responsável • Estimar tempo de realização da manutenção indicando na QM a data limite para conclusão
Ferramenta: • Completar Registro da QM no Jira com os atributos de contexto • Mover QM para a guia da próxima fase
Saídas: • Questão de Manutenção com registro completo
Observações: • N/A

Descrição das Atividades para Documentação

Fase: • 2-Projetar melhorias

Atividade: **Identificar funcionalidades da aplicação**
Descrição: Mapear funcionalidades da aplicação
Pré-condições: • Manutenção categorizada como Documentação
Entradas: • Código do servidor da aplicação
Tarefas: • Analisar o código da API identificando as funcionalidades/métodos presentes
Ferramenta: • N/A
Saídas: • Relação das funcionalidades da aplicação
Observações: • N/A

Fase: • 2-Projetar melhorias
Atividade: **Identificar funcionalidades ausentes na documentação**
Descrição: Mapear funcionalidades que precisam ser documentadas
Pré-condições: • N/A
Entradas: • Relação das funcionalidades da aplicação • Documentação da aplicação
Tarefas: • Comparar a documentação atual com as funcionalidades identificadas anteriormente
Ferramenta: • N/A
Saídas: • Relação das funcionalidades ausentes na documentação
Observações: • N/A

Fase: • 2-Projetar melhorias
Atividade: **Identificar instruções de usabilidade ausentes na documentação**

Descrição: Identificação de instruções de usabilidade e configuração do ambiente de desenvolvimento que estão ausentes na documentação
Pré-condições: • N/A
Entradas: • Documentação da aplicação
Tarefas: • Seguir as instruções presentes na documentação atual para identificar passos ausentes
Ferramenta: • Mover QM no Jira para próxima fase
Saídas: • Lista de instruções de usabilidade e configuração a serem incluídas
Observações: • N/A

Fase: • 3-Implementar e testar
Atividade: **Atualizar documentação**
Descrição: Implementação da atualização da documentação com adição e atualização das funcionalidades necessárias bem como instruções de usabilidade e configuração do ambiente de desenvolvimento
Pré-condições: • N/A
Entradas: • Documentação da aplicação • Lista de instruções de usabilidade e configuração a serem incluídas • Relação das funcionalidades ausentes na documentação
Tarefas: • N/A
Ferramentas: • N/A
Saídas: • Documentação atualizada
Observações: • N/A

Fase: • 3-Implementar e testar
Atividade: **Validar documentação**
Descrição: Validação da documentação seguindo as informações presentes na mesma após atualização

Pré-condições: • N/A
Entradas: • Documentação atualizada
Tarefas: • N/A
Ferramentas: • Mover QM no Jira para próxima fase
Saídas: • Documentação validada
Observações: • N/A

Fase: • 4-Implantar e monitorar
Atividade: **Atualizar documentação**
Descrição: Atualização da documentação no repositório da aplicação na branch principal
Pré-condições: • N/A
Entradas: • Documentação validada
Tarefas: • N/A
Ferramentas: • Mover QM no Jira para guia de Pronto • Finalizar QM
Saídas: • Repositório atualizado com nova documentação
Observações: • N/A

Descrição das Atividades para Refatoração, Correção de Bugs, Novos Requisitos e Mudança de Requisitos

Fase: • 2-Projetar melhorias
Atividade: **Análisar Logs**

Descrição: Rastreio da origem e identificação do bug através da análise de logs da aplicação e mensagens de erro ao executar a funcionalidade
Pré-condições: • Manutenção categorizada como Correção de Bugs
Entradas: • Logs da aplicação • Código da aplicação
Tarefas: • Identificar origem do bug para correção
Ferramentas: • N/A
Saídas: • Arquivo/Módulo que precisa de correção identificado
Observações: • N/A

Fase: • 2-Projetar melhorias
Atividade: **Identificar módulos afetados**
Descrição: Identificação dos módulos do sistema que são/serão afetados
Pré-condições: • Manutenção categorizada como Refatoração, Correção de Bugs, Novos Requisitos ou Mudança de Requisitos
Entradas: • Código da aplicação
Tarefas: • Mapear os arquivos que precisam ser alterados
Ferramentas: • N/A
Saídas: • Relação dos arquivos que precisam ser alterados
Observações: • N/A

Fase: • 2-Projetar melhorias
Atividade: **Identificar nível de dependência entre os módulos**
Descrição: Identificação das dependências do alvo da alteração com os demais módulos para maior clareza do risco e impacto da manutenção

Pré-condições: • N/A
Entradas: • Relação dos arquivos que precisam ser alterados • Código da aplicação
Tarefas: • Identificar o grau de dependência do(s) arquivo(s) alvo da manutenção com demais módulos
Ferramentas: • N/A
Saídas: • Relação dos arquivos que precisam ser alterados com níveis de impacto
Observações: • N/A

Fase: • 2-Projetar melhorias
Atividade: **Priorizar Implementação**
Descrição: Priorização das tarefas com base no nível de dependência entre o arquivo alvo e os demais módulos partindo dos mais acoplados para os mais simples
Pré-condições: • N/A
Entradas: • Relação dos arquivos que precisam ser alterados com níveis de impacto
Tarefas: • N/A
Ferramentas: • Mover QM no Jira para próxima fase
Saídas: • Relação dos arquivos que precisam ser alterados priorizada por nível
Observações: • N/A

Fase: • 3-Implementar e testar
Atividade: **Iniciar branch**
Descrição: Iniciar branch no repositório da aplicação para implementação da solução
Pré-condições: • N/A
Entradas: • Repositório da aplicação

Tarefas: • N/A
Ferramentas: • N/A
Saídas: • Branch criada no repositório
Observações: • Seguir padrão de nomenclatura de branch: "natureza da implementação/objetivo da atividade" Ex: feature/autocomplete; fix/userRegister; refact/getFungis

Fase: • 3-Implementar e testar
Atividade: **Desenvolver solução**
Descrição: Implementar solução da manutenção na branch criada anteriormente
Pré-condições: • N/A
Entradas: • Branch criada no repositório • Código da aplicação
Tarefas: • N/A
Ferramentas: • N/A
Saídas: • Código da aplicação atualizado com as mudanças
Observações: • N/A

Fase: • 3-Implementar e testar
Atividade: **Realizar testes**
Descrição: Testar a funcionalidade alvo da manutenção bem como os módulos/arquivos que sofreram modificações para garantir a suficiência da implementação
Pré-condições: • N/A
Entradas: • N/A
Tarefas: • N/A

Ferramentas: • Mover QM para próxima fase no Jira
Saídas: • Relação dos testes de funcionalidades executados
Observações: • N/A

Fase: • 3-Implementar e testar
Atividade: **Iniciar Pull Request**
Descrição: Testar a funcionalidade alvo da manutenção bem como os módulos/arquivos que sofreram modificações para garantir a suficiência da implementação
Pré-condições: • Equipe de manutenção adotou Code Review (revisão de código)
Entradas: • Branch criada para execução da QM
Tarefas: • N/A
Ferramentas: • Completar registro da QM com atributo do link/url do Pull Request criado
Saídas: • Pull Request criado no repositório do projeto
Observações: • N/A

Fase: • 3-Implementar e testar
Atividade: **Realizar Code Review**
Descrição: Realizar a revisão de código no pull request da QM
Pré-condições: • N/A
Entradas: • Pull Request criado no repositório do projeto • Link do Pull Request criado
Tarefas: • N/A
Ferramentas: • N/A

Saídas: • Pull Request aprovado
Observações: • N/A

Fase: • 3-Implementar e testar
Atividade: **Atualizar aplicação em desenvolvimento**
Descrição: Realizar o merge das alterações realizadas e aprovadas no pull request à branch de desenvolvimento
Pré-condições: • N/A
Entradas: • Pull Request aprovado
Tarefas: • N/A
Ferramentas: • N/A
Saídas: • Código da aplicação atualizado na branch de desenvolvimento
Observações: • N/A

Fase: • 4-Implantar e monitorar
Atividade: **Atualizar aplicação**
Descrição: Realizar o merge da branch de desenvolvimento ao final da fase de implementação e testes com a versão de produção da aplicação no repositório
Pré-condições: • N/A
Entradas: • Relação dos testes de funcionalidades executados • Código da aplicação atualizado com as mudanças
Tarefas: • N/A
Ferramentas: • N/A
Saídas: • Código da aplicação atualizado na branch de produção

Observações: • N/A

Fase: • 4-Implantar e monitorar
Atividade: **Publicar alterações**
Descrição: Publicar aplicação atualizada em ambiente de produção
Pré-condições: • N/A
Entradas: • Código da aplicação atualizado na branch de produção
Tarefas: • N/A
Ferramentas: • N/A
Saídas: • Nova versão do sistema em execução com as correções implementadas
Observações: • N/A

Fase: • 4-Implantar e monitorar
Atividade: **Monitorar**
Descrição: Monitorar aplicação atualizada após publicação para observar e garantir funcionamento correto da solução implementada no cenário de produção
Pré-condições: • N/A
Entradas: • Nova versão do sistema em execução com as correções implementadas
Tarefas: • N/A
Ferramentas: • Mover QM no Jira para guia de Pronto • Finalizar QM
Saídas: • Nova versão do sistema em execução com as correções implementadas

Observações: • N/A

Descrição das Atividades para Dockerização

Fase: • 2-Projetar melhorias
Atividade: **Análise de dependências**
Descrição: Analisar e entender como a tecnologia de implementação da aplicação lida com as dependências e como as mesmas podem ser instaladas no ambiente de contêiner
Pré-condições: • Manutenção categorizada como Docker
Entradas: • Código da aplicação
Tarefas: • N/A
Ferramentas: • N/A
Saídas: • Instruções para obter as dependências
Observações: • N/A

Fase: • 2-Projetar melhorias
Atividade: **Análise de serviços**
Descrição: Analisar os serviços que precisam ser acoplados e/ou isolados em seus próprios contêineres
Pré-condições: • N/A
Entradas: • Código da aplicação • Arquitetura da aplicação
Tarefas: • N/A
Ferramentas: • N/A

Saídas: • Relação de serviços presentes na aplicação
Observações: • N/A

Fase: • 2-Projetar melhorias
Atividade: **Análise de volumes**
Descrição: Analisar e entender quais volumes precisam ser criados para garantir persistência dos dados e acesso da aplicação a arquivos gerados por algum dos serviços
Pré-condições: • N/A
Entradas: • Código da aplicação
Tarefas: • N/A
Ferramentas: • N/A
Saídas: • Relação dos diretórios e volumes a serem mapeados
Observações: • N/A

Fase: • 2-Projetar melhorias
Atividade: **Definição do cenário**
Descrição: Definir o cenário da conteinerização com base na análise dos serviços • Cenário 1:Interação entre diversos contêineres; Cada serviço da aplicação isolado em um contêiner. • Cenário 2: Aplicação completa pode ser replicada em um único contêiner.
Pré-condições: • N/A
Entradas: • Código da aplicação • Relação de serviços presentes na aplicação
Tarefas: • N/A

Ferramentas: • Mover QM para próxima fase no Jira
Saídas: • Arquitetura da aplicação para contêineres
Observações: • N/A

Fase: • 3-Implementar e testar
Atividade: **Escrita Dockerfile**
Descrição: Escrita do arquivo Dockerfile, se necessário, com base na análise das dependências realizada
Pré-condições: • N/A
Entradas: • Arquitetura da aplicação para contêineres • Instruções para obter as dependências
Tarefas: • N/A
Ferramentas: • N/A
Saídas: • Dockerfile
Observações: • N/A

Fase: • 3-Implementar e testar
Atividade: **Escrita compose**
Descrição: Escrita do arquivo docker-compose, se necessário, com base na análise de serviços e de volumes
Pré-condições: • N/A
Entradas: • Arquitetura da aplicação para contêineres • Relação dos diretórios e volumes a serem mapeados • Relação de serviços presentes na aplicação

Tarefas: • N/A
Ferramentas: • N/A
Saídas: • docker-compose.yml
Observações: • N/A

Fase: • 3-Implementar e testar
Atividade: **Build do contêiner**
Descrição: Realizar build do contêiner para atestar a execução correta da aplicação e identificar possíveis erros como problemas na versão da imagem ou falhas no gerenciamento das dependências
Pré-condições: • N/A
Entradas: • Dockerfile • docker-compose
Tarefas: • N/A
Ferramentas: • N/A
Saídas: • Contêiner(es) da aplicação *buildados*
Observações: • N/A

Fase: • 3-Implementar e testar
Atividade: **Testes**
Descrição: Realizar testes das funcionalidades da aplicação em contêiner para garantir o funcionamento e interação corretos entre os serviços
Pré-condições: • Contêiner em execução
Entradas: • Contêiner(es) da aplicação

Tarefas: • N/A
Ferramentas: • N/A
Saídas: • Relação dos testes de funcionalidades executados
Observações: • N/A

Fase: • 4-Implantar e monitorar
Atividade: **Configurar rede**
Descrição: Adicionar configuração ao contêiner para execução em rede docker criada no ambiente de produção
Pré-condições: • Aplicação disponível no ambiente de produção
Entradas: • docker-compose.yml
Tarefas: • N/A
Ferramentas: • N/A
Saídas: • docker-compose.yml atualizado com rede docker do ambiente
Observações: • N/A

Fase: • 4-Implantar e monitorar
Atividade: **Configurar proxy**
Descrição: Configurar acesso a aplicação por proxy no ambiente de produção
Pré-condições: • Aplicação configurada na rede docker adequada
Entradas: • docker-compose.yml ajustado ao ambiente

Tarefas: • N/A
Ferramentas: • Mover QM no Jira para guia de Pronto • Finalizar QM
Saídas: • Arquivo de configuração do proxy • Aplicação acessível externamente via url
Observações: • N/A

Melhorias

A última fase do processo compreende a melhoria contínua, ao final do processo, ou seja, a finalização de uma questão de manutenção deve-se realizar uma revisão das estratégias e etapas para que o processo seja adaptado e melhorado para as próximas iterações de manutenção. Em um contexto de equipe de manutenção podem ser trabalhados conjuntos, agrupamentos de QMs dentro de um período definido, dessa forma a revisão da adoção e implementação do processo ao longo do período pode ser trabalhada em uma reunião a fim de adequar o processo ao contexto e condições de trabalho da equipe.